Web API 品質モデルの提案とその定量評価

山本 里枝子[†1] 大橋 恭子[†1] 福寄 雅洋[†1] 木村 功作[†1]
関口 敦二[†1] 梅川 竜一[†1] 上原 忠弘[†1] 青山 幹雄[†2]

概要：クラウドの普及に伴い様々なサービスを Web API で提供，利用するソフトウェア開発形態が増加している．そのため，Web API の品質がそれを利用したアプリケーション開発の生産性と品質に大きな影響を及ぼすことが明らかになってきたが，品質モデルは確立されているとはいえない．特に，従来のシステム内 API と異なり，Web API はリモートで実行され，ユーザと独立に変更されるなど，ソフトウェア工学の新たな問題を提起しており，最近提供が始まったエンタープライズ API の利用において大きなリスクとなっている．このような問題に対して，本稿では，Web API の品質モデルを提案する．Web API を利用するアプリケーション開発者に対する品質モデルの主要品質特性として，API ユーザビリティの習得容易性と変更の安定性の概念とその評価尺度を提案する．提案した品質特性を実際の Web API に適用し，API ユーザビリティの習得容易性の測定値が開発者による定性的な評価と同じ傾向を示すこと，及び，変更の安定性が Web API の成熟度と同じ傾向を示すことから，その有効性を確認した．

キーワード：Web API, REST，品質モデル，品質特性, API ドキュメント，習得容易性，安定性

1. はじめに

近年，Web API が企業情報システム開発のコア資産となっている[1][2]．Web API はインターネットを介した Web サービスを提供する手段であり，REST (Representational State Transfer)アーキテクチャスタイルに準拠する[3]．Web API のグローバルなディレクトリ Programmable Web[4]には 2018 年 7 月現在で約 20,000 の Web API が登録され，今後も急速に増加すると見込まれている．さらに，Web API を基礎とするソフトウェアとビジネスのエコシステムが国内外で成長している[1][5]．また，企業がエンタープライズ API として戦略的に Web API をパートナやコミュニティに公開し，新しいサービスやアプリケーションの開発を促し，API ビジネスを展開する動きがある[2]．このように Web API は今後の企業情報システム開発のプラットフォームとして戦略的な研究開発が必要である．

しかし，企業情報システム開発に Web API を利用するには多くの課題がある[6][7]．Web API は REST[3]を基礎としてそのインタフェース定義は実装言語と独立なリソースのテキスト表現(Representation)となるため，非形式的で，かつ，標準化もされていない．実際，多くのインタフェース定義の記述形式は個別で厳密性に欠けている．企業システム開発に Web API を利用するためには品質保証は不可欠であるが，Web API はそのユーザとは独立にリモートで開発，保守され実行されることから，従来のシステム内の API(以下，システム API と呼ぶ)とは異なる問題を提起している．さらに，オープンなサービスとして公開され，アジャイル開発，DevOps などの新たな開発，提供形態の拡大に伴い，Web API が短期間に，あるいは，継続的に変更される場合も発生している[8][9]．このような背景のもとに Web API に関するソフトウェア工学の研究が必要とされているが，従来のシステム API と比較し，その研究は極めて少なく萌芽的な段階にある[10]．

本稿では，企業情報システム開発を対象に，Web API の品質問題に焦点を当てる．そのため，Web API が持つ従来のシステム API と異なる性質に着目し，品質モデルの定義とその具体的な品質特性を明らかにする．さらに，その品質特性を評価するための尺度と測定方法を提案する．

本稿の構成は以下の通りである．2 節で研究課題，3 節で関連研究を述べ 4 節でアプローチを説明する．5 節で Web API 品質モデルを提案し，6 節で実際の Web API に適用した結果を述べた後，7 節で品質モデルの有効性を考察する．

2. 研究課題

企業情報システム開発で Web API の利用を促進するためには，Web API の品質を定量的に測定する必要がある．Web API は従来のシステム API と異なる以下のような特徴を持つため，その違いを考慮して議論する必要がある．

- インタフェース定義：従来のシステム API は，その実装言語でインタフェースを定義することから，インタフェース定義を生成するツールも提供されている．しかし Web API のインタフェース定義は，実装言語とは独立に REST の原則[3]に従うリソースの表現として定義される．そのためツール化されておらず，インタフェース定義が人手に依存している．さらに，そのリソース表現は，同一リソースであっても，JSON や XML など複数の形式を取り得て，自己記述メッセージ(Self-descriptive Message)として交換される．このため，型チェックなどが適用できない．このようなインタフェース定義の品質モデルは未確立である．
- 実行時の独立した進化：Web API はインターネットを介した Web サービスを提供する手段である．従来のシステム API がそのユーザの計算機資源の中に取り込んだ

†1 (株)富士通研究所　Fujitsu Laboratories Ltd.
†2 南山大学　Nanzan University,

ライブラリ等をアクセスするのに対し，Web API はユーザの計算機資源とは異なる環境で開発，修正され，遠隔にサービスとしてアクセスされる．そのため，ユーザと独立して，機能の変更，追加が可能である．一方，利用できていた Web API が，ユーザが知らない間に変更され，その Web API を使ったアプリケーションが動作しなくなる，という問題にもつながる[9].

我々は，システム API とは異なる上記のような特徴を持つ Web API を利用したアプリケーション開発において，アプリケーションの品質確保を目的に，Web API の品質問題に取り組む．そのため，以下の研究設問を設定する．

RQ 1: WebAPI の品質モデルは何か?

Web API を利用したアプリケーションの品質を確保するために，これまでのシステム API と異なる Web API の特徴を捉えた，Web API の品質モデルを定義する．Web API のインタフェース定義が実装言語と独立なリソースのテキスト表現であること，ユーザとは独立した開発サイクルが可能であること，等の品質に関わる新たな課題を検討し，品質モデル，品質特性を定義する．

RQ 2: 提案する品質特性の尺度とその測定方法は何か?

Web API の特徴を捉えた品質特性の尺度と実測可能な測定方法を明らかにする．

RQ3: 提案方法は実際の Web API に有効か?

実際に利用されている Web API を対象に，提案した品質モデルとその尺度，測定方法を適用し，妥当性を実証する．

3. 関連研究

3.1 Web API の急速な普及とその課題

Web API の急速な普及に伴い，その価値の認識[2][5]と共にその技術課題も明らかになってきた[7]．しかし，従来のシステム API と比較し Web API に関する研究は極めて少なく，萌芽的段階にあるといえる．さらに，従来主としてスマートフォンアプリケーションや Web アプリケーション向けであった Web API に対してエンタープライズ API と呼ばれる企業情報システム開発や B2B 向けの Web API が提供されるようになり，Web API の品質や Web API を用いた企業情報システム開発が重要な課題となっている[10].

3.2 API の品質とアプリケーション開発への影響分析

Web API の品質はその仕様記述ドキュメントの品質として捉えられている．Web API の品質は，それに関与するステークホルダのパースペクティブによる[11]．API 品質はそのユーザであるアプリケーション開発者に最も影響を及ぼすことから開発者のパースペクティブから評価すべきであると考えられている[6]．API 品質がそれを用いたアプリケーション開発の生産性に影響していることが実態調査と実証実験により明らかになっている．その中で API ユーザビリティの重要性が認識され，その研究が活発におこなわれてきた．ソフトウェア製品のユーザビリティの概念をシステム API へ拡張した API ユーザビリティの概念が提唱された[12]．さらに，この概念を Web API に適用した API ユーザビリティが最近提唱されている[13]．API ユーザビリティがアプリケーション開発の生産性に及ぼす影響は開発者へのアンケート調査[14]や実証実験[15]によって明らかになっている．

しかし，これらの研究においては，ユーザビリティを含む幾つかの品質特性が示されているに留まり，Web API の品質特性に関する包括的な定義は未確立である．

3.3 API 仕様記述における例示の効果

システム API の利用において，その仕様記述に例示を含むことの有効性が示されている[16]．この成果を Web API に適用し，API 仕様記述に例示を含む効果を開発比較実験により実証している[15] ．また，アンケート調査によって適切な例示の必要性も示されている[17]．これらの結果から Web API 仕様記述のスタイルガイドラインの検討も提案されている[18]．しかし，従来の研究では例示の有無に留まり，その構造や内容などに関する議論は見られない．

3.4 Web API の進化の問題

システム API に対して Web API が異なる本質的特性の一つに “実行時の独立した進化” がある．すなわち API コンシューマが API 利用時に API プロバイダが API を独立に変更，削除する可能性があることである．これは Web サービス共通の特性として知られているが，サービス中断などの深刻な問題を引き起こすリスクとなる[19]．さらに，API が変更されるとそれを利用するアプリケーションも変更せざるをえない状況がでてくる．例えば，Li らは 226 件の API の変更を 16 のパターンに分類し，その中に従来のシステム API にない新たな変更パターンを 6 つ指摘した[20]．この研究を発展させ，Wang らは StackOverflow 上での質疑から，21 変更パターンを特定し，その中で，7 つが新たなパターンとしている[9]．また，約 82%の変更が改版の途中で行われていることを指摘し，変更が改版によらず常時行われていることも指摘している．近年のアジャイル開発や DevOps の普及により，この傾向は一層顕著になる可能性がある．

4. アプローチ

一般に，品質は ISO 29148[21]で規定するようにステークホルダ毎に異なる．我々は，API の品質も同様にそれに関与する次の様なステークホルダにより異なると考える．

(1) API プロバイダ: Web API を実装し，提供するバックエンドサービスの提供者.
(2) API コンシューマ: Web API を用いたアプリケーション／プロダクトの開発者.
(3) アプリケーションユーザ: Web API を利用したアプリケーションのユーザ

本稿では，Web API の使用を伴うような DX (Developer

eXperience)の改善を目的に、API コンシューマの視点からの品質に焦点をあてる．

表 1 にステークホルダの中で, API の直接的, あるいは, 間接的なユーザの視点から整理した API 品質特性の一部を示す．従来，個別に提案されてきた Web API のユーザビリティなどの品質特性[11][12][13]の概念を整理し，製品品質標準 ISO 25010[22]を拡張して分類したものである．

表 1　Web API 品質特性

ステークホルダ	対象分類	品質特性分類	品質特性
API コンシューマ	API 表現	ユーザビリティ(Usability)	習得容易性(Learnability), メンタルモデル適合性(Matching mental Model) 生産性(Productivity) エラー防止性(Error-Prevention)
		理解容易性(Understand-ability)	単純性(Simplicity), 一貫性(Consistency) 表現性(Expressiveness)
	API 内容	進化可能性(Evolvability)	拡張性(Extensibility), 安定性(Stability)
アプリケーションユーザ	アプリ内容	ユーザビリティ(Usability)	生産性(Productivity) エラー防止性(Error-Prevention)

ここで，対象分類とは品質を評価する対象の分類である．API では，その表現(Description)と内容(Internal)に分けられる．さらに，品質特性分類は ISO 25030[23]で規定されている製品品質の外部品質と利用品質の概念を拡張したものである．例えば，生産性であっても，開発者にとっては Web API を利用した開発の生産性となり，アプケーションユーザにとってはそれを実行して目的を達成する生産性を意味し，異なる定義となる．

以降，Web API と従来のシステム API の特徴の差を捉えた品質特性に着目する．本稿ではその中で，Web API を用いたアプリケーション開発の初期に，API コンシューマにとって重要となる，以下の二つの品質特性を扱う．

- 習得容易性 (Learnability) : 習得容易性を, ある特定のユーザがある特定の目標を達成するために Web API を使用することによる有効性, 効率性, リスクと満足度の程度，と定義する．ISO 25010 のソフトウェア製品品質の習得性の拡張であり，ユーザビリティの一特性である．3.2 で述べたように，システム API での API ユーザビリティの重要性は実証されており，また，Web API でも API ユーザビリティの重要性が認識されつつあるため，ユーザビリティを対象とする．システム API がその実装言語でインタフェースを定義するので一意の理解となるのに対し，Web API のインタフェース定義は実装言語と独立に REST の原則に従うリソースの表現として定義し, 同一リソースでも複数の表現をとり得る．この性質の違いに着目して，ユーザビリティの中でもその起点となる習得容易性を本稿の対象とする．
- 安定性 (Stability) : 安定性を，Web API のインタフェースがある時間にわたって変化する度合い，と定義する．これは，Web API コンシューマが変更に対応するための開発期間への制約となる．この定義は、ライブラリの安定性[25]を Web API に拡張したものである。3.4 で述べたように, API が変更されるとそれを利用するアプリケーションも変更せざるをえない状況が出てくる. Web API はシステム API と異なり, API コンシューマと独立にリモートで開発されて，“実行時の独立した進化”をする特徴を持つ．これは，企業情報システムに大きなリスクとなることから，開発の初期段階で Web API の変更を予測する必要がある．このために，本稿では安定性を対象とする．

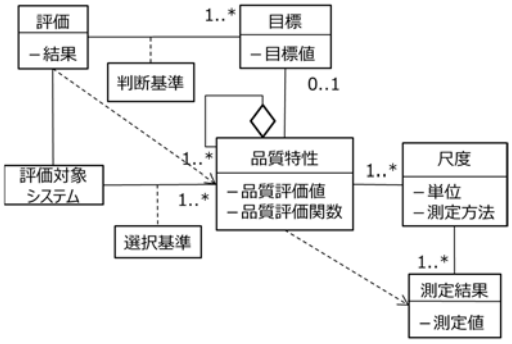

図 1　品質評価メタモデル

5. API の品質モデル

5.1 メタモデル駆動型 Web API 品質モデル

関連研究で述べたように，Web API を用いる開発の広まりとともに，本稿を含む様々な品質特性の議論が期待される．そのため，Web API の品質モデルの全体像として，概念，スコープ，語彙を定義するためのメタモデルを最初に定義する．次に，各 Web API 品質特性の品質モデルをメタモデルのインスタンスとして定義する．メタモデルは Web API 品質モデルの基礎となり，モデルの一貫性を保証するために用いる．

5.2 メタモデル

ソフトウェア開発の視点から捉えた品質特性とその測定の共通認識の基礎として，ソフトウェアの汎用的な品質評価のメタモデルをまず定義する．ISO 15939 測定情報モデル[24] (プロジェクトで必要な情報を測定可能なプロセスやプロダクトと関係づけるモデルの定義)を参照し, 品質特性とその評価に必要な要素のモデルを図 1 に示す．

品質特性は階層的に構成しその最下層の品質特性が測定対象となるように定義する．評価対象システムは必要とする品質特性を選択し，測定結果と目標値を比較して品質を評価する．

5.3 習得容易性

5.3.1 習得容易性のモデル

Web API を用いたアプリケーション開発では，API コンシューマは使用する Web API に対してある程度習熟してい

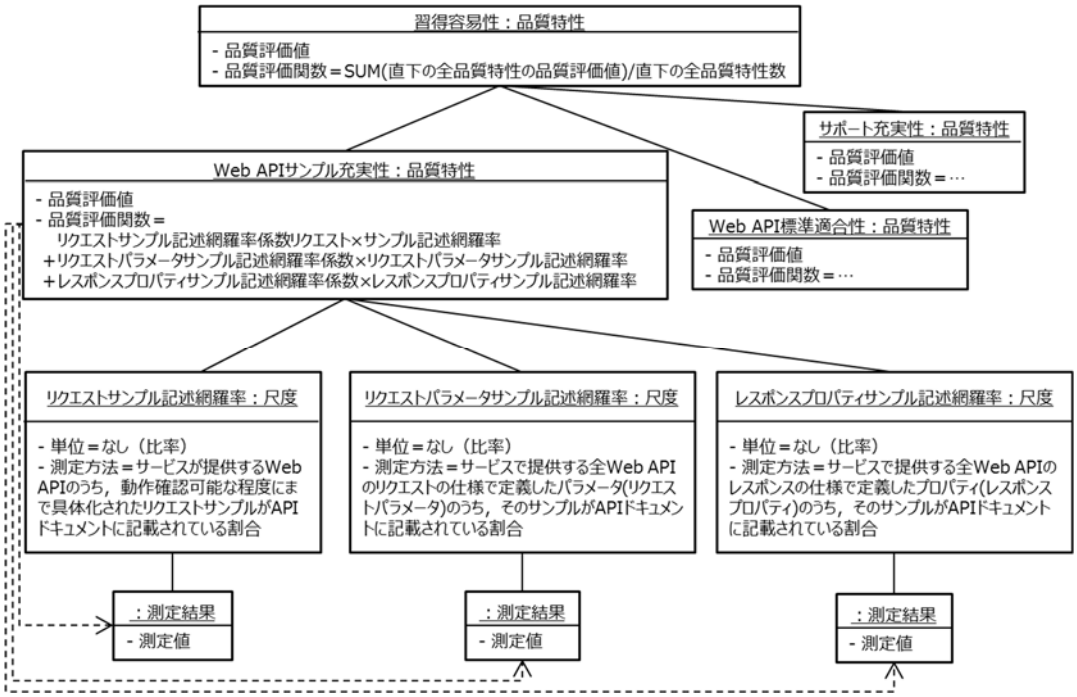

図 2 サービス評価品質モデルの一部(習得容易性)

る必要がある．そのため，開発に利用する Web API を新たに選択する場合，習得しやすいものを選択することが望まれる．我々は，Web API の習得しやすさ(以後，習得容易性と呼ぶ)に注目して品質特性を具体化した．前節に示したメタモデルの具体化に際しては，開発初期の動作環境が整備される前であっても利用可能な情報，例えば API ドキュメントを用いた尺度を定義することを目指した．

Web API の習得容易性の品質モデルを図 2 に示す．この品質モデルでは，習得容易性を以下の 3 つの品質特性に詳細化した．各品質特性の概要と品質モデルの中で取り上げた理由を以下に示す．

(1) WEB API サンプル充実性：Web API がその仕様だけでなくサンプルでも示されている程度．API コンシューマが Web API の選択や習得し始めようとした際にまず参照するのがサンプルだと想定したためである．

(2) Web API 標準適合性：Web API が設計原則の標準である REST [3]に則っていること．標準設計原則に則った Web API であれば，その仕様の理解が容易で，開発にも利用しやすいと考える．

(3) サポート充実性：API コンシューマに対するサポートが充実している程度．Web API に限らず何かを習得しようとする際には，疑問点や意図したとおりにならないことが発生する．そのような場合に効率的に解決できることはアプリケーション開発に重要であるため，本品質特性を取り上げた．ここで，サポートとは API コンシューマ向けの情報提供サイトや FAQ 等である．

これら 3 つの品質特性のうち，以降では「(1) WEB API サンプル充実性」について議論する．この品質特性を取り上げた理由は，品質特性の達成度合いの機械的な判断に最も適していると判断したからである．

5.3.2 習得容易性の尺度

3.3 で述べたように API のサンプルが習得容易性に寄与することが Uddin ら[14]や Sohan ら[15]により実証されている．我々もサンプルの重要性を認識し，前節の品質特性「WEB API サンプル充実性」に注目して 3 つの尺度を定義する (表 2)．尺度は次の二つの観点から整理した．

- 対象：サンプルが API のリクエストとレスポンスのいずれを対象としているか
- 粒度：サンプルが，API のリクエストのサンプルであるか，あるいはリクエストやレスポンスを構成するパラメータプロパティのサンプルであるか

以降，リクエストを Req.，レスポンスを Res.と略す．

表 2 習得容易性の尺度

		粒度	
		全体	構成要素
対象	Req.	(1) リクエストサンプル記述網羅率 (ReqSampleCoverage)	(2)リクエストパラメータサンプル記述網羅率 (ReqParamCoverage)
	Res.	－	(3)レスポンスプロパティサンプル記述網羅率 (ResPropCoverage)

表中の「リクエストサンプル」とは，Web API の動作確認をするために最低限必要な HTTP メソッド名，エンドポイントのパス，全必須パラメータと任意個のオプショナルなパラメータのサンプルの組み合わせである．

これら習得容易性の 3 つの尺度の定義を示す．

(1) リクエストサンプル記述網羅率(ReqSampleCoverage)：

API コンシューマによる動作確認が可能な程度に具体化したリクエストのサンプルが提供されている程度を表す. 定義を式(1)に示す. 値域は 0 から 1. 全ての Web API に対してリクエストのサンプルが存在すると 1, 全く存在しないと 0 となる.

NumOfAPIs：サービスが提供する Web API の総数

NumOfAPIsWithSample：サービスで提供している全 Web API のうち，そのままで動作確認可能なリクエストサンプルを提供する API の数

$$\text{ReqSampleCoverage} = \frac{NumOfAPIsWithSample}{NumOfAPIs} \quad (1)$$

(2) リクエストパラメータサンプル記述網羅率 (ReqParamCoverage)：リクエストに用いるパラメータの仕様に対して，その使い方を理解するためのサンプルが提供されている程度を表す．定義を式(2)に示す．値域は 0 から 1．全パラメータにサンプルが存在すると 1，全く存在しないと 0 となる．

NumOfParams : サービスで提供する全 Web API のリクエストの仕様で定義したパラメータの総数

NumOfParmsWithSample：サービスで提供する全 Web API の全リクエストパラメータの中で，そのサンプルを API ドキュメントに示したパラメータの数

$$\text{ReqParamCoverage} = \frac{NumOfParmsWithSample}{NumOfParams} \quad (2)$$

リクエストのパラメータの仕様とそのサンプルの例を図 3 と図 4 に示す．また，図中の Web API の例を用いてリクエストサンプル記述網羅率とリクエストパラメータサンプル記述網羅率の計算方法を示す.

Name	Type	Description
discountCode	string	The discount code on the coupon.

(a) パラメータの仕様の例

```
curl -X PATCH –H “token=<TOKEN>” ¥
    -H “Content-Type: application/json” ¥
                -d ‘{
        “discountCode”: “XYZ”
    }’ https://api.abcompany.com/v1.1/user
```

(b) リクエストサンプルの例

図 3 Web API の定義例 1

Name	Type	Description
deliveryDate(optional)	string	The specified date for the delivery.
deliveryTime(optional)	string	The specified time for the delivery.

(a) パラメータの仕様の例

```
curl –X POST -H “token=<TOKEN>” ¥
    - ‘Accept-Language: en_US’ ¥
    -H “Content-Type: application/json” ¥
                -d ‘{
    }’ https://api.abcompany.com/v1.1/delivery
```

(b) リクエストサンプルの例

図 4　Web API の定義例 2

まず，リクエストサンプル記述網羅率の計算例を示す. Web API は 2 個なので，NumOfAPIs は 2 である．各図の(a)に示した Web API の仕様に対して(b)に示したサンプルがあるので，開発されるアプリ毎に異なる部分，図中の <TOKEN>部分, 以外はそのまま使用して動作確認が可能なサンプルは 2 個ある. 従って NumOfAPIsWithSample は 2 である．NumOfAPIs と NumOfAPIsWithSample から，ReqSampleCoverage の値として 1 が得られる.

次に，リクエストパラメータサンプル記述網羅率の計算例を示す. 図 3(a)には 1 つのパラメータ” discountCode”, 図 4(a)には 2 つのパラメータ” deliveryDate”と” deliveryTime” の仕様が記述されている．2 つの図で合計 3 個のパラメータの仕様が記述されているので, NumOfParams は 3 である. 次に，図 3(b)にはパラメータ” discountCode”のサンプルが含まれているが，図 4(b)にはパラメータ” deliveryDate”と” deliveryTime”のサンプルは含まれていない．サンプルを持つパラメータは 1 個だけなので NumOfParmsWithSample は 1 である．NumOfParams と NumOfParmsWithSample から ReqParamCoverage の値として 1/3 が得られる.

(3) レスポンスプロパティサンプル記述網羅率 (ResPropCoverage) : レスポンスを構成するプロパティの仕様を理解するためのサンプルが提供されている程度を表す．定義を式(3)に示す．値域は 0 から 1．全プロパティにサンプルが存在すると 1，全く存在しないと 0 となる.

NumOfProps : サービスが提供する Web API のレスポンスで，仕様として定義したプロパティの総数

NumOfPropsWithSample : サービスで提供する全 Web API の全レスポンスプロパティの中で，そのサンプルを API ドキュメントで示したレスポンスプロパティの数

$$\text{ResPropCoverage} = \frac{NumOfPropsWithSample}{NumOfProps} \quad (3)$$

本尺度の計算に必要なデータは，ReqParamCoverage と同様にして API ドキュメントから取得する.

5.4 安定性

5.4.1 安定性のモデル

3.4 で述べたように Web API は変更が改版によらず常時行われているため，変更が発生した場合の追従に想定外の工数が発生する可能性がある．そのため，Web API を利用する前に保守の期間や工数を予測できることが重要である. このための品質特性として Web API の安定性(Stability)を定義する．従来のシステム API の安定性は，Raemaekers らにより「ソフトウェアライブラリのユーザが対応を要するかもしれない，時間の経過に伴い積み重なっていくインタフェースや実装の変更の度合い」として定義され，削除メソッド数，既存メソッド内の変更割合，新旧メソッドの変更割合比，新メソッドの割合の 4 つの指標が提案されている[25]．それらの指標値の算出には実装上の API の変更を

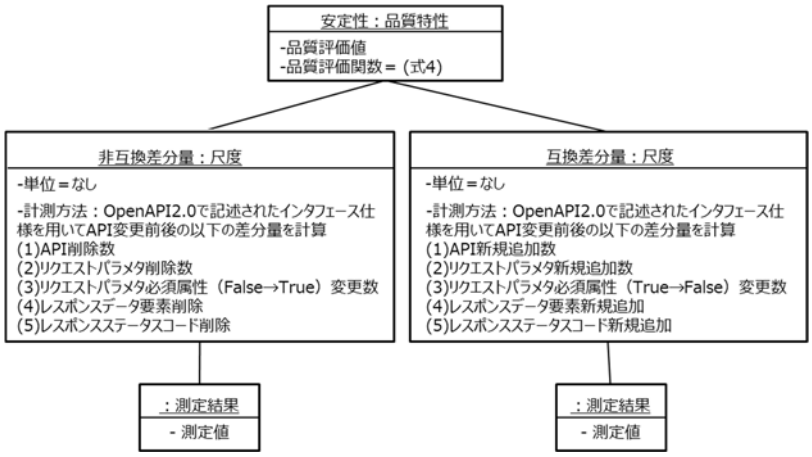

図 5 サービス評価品質モデルの一部(安定性)

抽出する必要があるが 2 節で述べたように Web API のインタフェース定義は実装言語とは独立に REST の原則に従うリソースの表現として定義されるため，従来のシステム API に対する測定方法はそのままでは適用できない．そこで Li ら，Wang らによって提案された Web API の変更のパターン分類[9][20]と同様に，Web API の構成要素をもとに Web API の変更を評価する方法を提案する．そして Web API に関しても，Raemaekers らの安定性の定義と同様に Web API ユーザである API コンシューマが必要となるであろう API 変更への対応の度合いを安定性として定義する．定義において，API コンシューマに必要となる対応が少ない場合は安定性が高く，多い場合は安定性が低いとする．そこで Web API の変更数を互換性有無の観点で区別して安定性の算出に利用する．特に，互換性のない変更は安定性の低下が大きく，互換性のある変更は低下が小さくなるように定義する．互換性のある変更は API コンシューマに影響を与えないと考えることもできるが，インタフェース上は変更がない場合においても振る舞いが異なるために API コンシューマの対応が必要な場合があるため，本定義では安定性の低下をもたらすものとした．

以上にて説明した安定性の品質モデルを図 5 に示す．

5.4.2 安定性の尺度

Web API の変更量を算出するために，変更前後の 2 つの Web API の差分を抽出する必要がある．Web API は従来のシステム API とは異なり，ネットワーク上に存在する機能(リソース)を表現した文字列によって呼び出す．このため，従来の構文解析による抽出方法の適用は困難であり，Web API のインタフェース仕様記述の差分抽出方法をとることが多い．本稿においてもインタフェース仕様である API リファレンスを利用して API の差分を抽出する方法を採用する．以降では安定性の品質モデルで取り上げた非互換差分量と互換差分量の算出方法について説明する．

(1) 非互換差分量

Web API に変更が発生した場合に API コンシューマが変更を必要とするような非互換な変更の要素数を非互換差分量と定義し，以下の測定値の総和とする．

a) API 削除数
b) リクエストパラメータ削除数
c) リクエストパラメータ必須属性(False→True)変更数
d) レスポンスデータ要素削除数
e) レスポンスステータスコード削除数

(2)互換差分量

Web API に変更が発生した場合に API コンシューマが変更を必要としないような互換性のある変更の要素数を互換差分量と定義し，以下の測定値の総和とする．

a) API 新規追加数
b) リクエストパラメータ新規追加数
c) リクエストパラメータ必須属性(True→False)変更数
d) レスポンスデータ要素新規追加数
e) レスポンスステータスコード新規追加数

ソフトウェアで変更が発生した箇所は，最初は不安定な状態だが変更を重ねると安定する傾向をとる性質がある. このような時間的変化の傾向が指標値に利用されることがある [26]．従来のシステム API の安定性についても「時間が経つにつれて変更の影響は小さくなる」と仮定し，変更発生からの時間を重みとして利用する．この仮定は Web API においても成立すると考え, 安定性の品質特性の算出に API 変更が発生してからの時間を重み $hw(s)$ として導入する．

また，5.4.1 で説明した API コンシューマへの影響の大きさは(2)互換差分量よりも(1)非互換差分量が大きくなる．このため, 安定性の品質評価関数において, (1)の重みを大きくするために，重みとして非互換差分量係数 k_i，互換差分量係数 k_c を導入する．

以上より, 安定性の品質特性の評価関数を(式 4)で定義する．値域は 0 から 1 であり，1 に近ければ安定しており，0 に近ければ安定していないことを示す．この評価値により保守工数の推定が可能となる．評価関数で用いる変数を表 3 に示す．

$$\text{安定性の品質評価関数} = \frac{1}{1+\sum_{s=1}^{|S|-1}(k_i I(s)+k_c C(s))hw(s)} \quad (4)$$

表 3 評価式(4)の変数定義

記号	説明
s	ソフトウェアの特定のスナップショットを示す番号, 1 から始まり過去にさかのぼるにつれて増加する
S	すべてのスナップショットの集合
k_i	非互換差分量係数
k_c	互換差分量係数
$I(s)$	非互換差分量(s+1 から s の間に発生した差分量)
$C(s)$	互換差分量(s+1 から s の間に発生した差分量)
$hw(s)$	時間係数 $\frac{1}{2^s}$

6. 品質モデルの定量評価

6.1 習得容易性の評価

習得容易性の尺度を表 4 に示す 4 つのサービスの Web API に対して測定した．測定したサービスの概要や特徴，測定方法，および測定結果を示す．

表 4：測定対象のサービス

サービス名(版)	概要や特徴
Uber (v1.2)	配車サービス向けのサービス
WordPress(V2)	Web ページやブログの作成を支援するサービス
OpenStack (Version Pike)	オープンソースで開発されているクラウドの IaaS 環境を構築するためのソフトウェアで，多くのコンポーネントから構成されている．本稿では，Ceilometer, Cinder, Designate, Glance, Heat, Keystone, Neutron, Nova, Swift のコンポーネントを利用した．外部に API を公開しているだけでなく，API 開発者も API を利用していることが特徴である
メディア処理 API(v1)	社内向けに公開されている画像や音声等の各種メディアを処理する Web API 群である．API ドキュメントは，Swagger UI で公開されている[27]．我々は，本 API 群の API プロバイダに対し，API コンシューマの立場で良いドキュメントとは何かについて指導を行っていた

各サービスに対し，以下の方法で尺度を測定した．

(1)尺度「リクエストサンプル記述網羅率」

Web 上に公開されている API ドキュメントを参照しながら人手で測定した. OpenAPI Specification (OAS) 2.0 形式[28]のファイルを入力とし，API ドキュメントを生成するツール[27]が公開されている．このツールを利用して API ドキュメントを公開する場合，ツール機能を用いてリクエストサンプルを生成できる．このようなツールで API ドキュメントを公開し，リクエストサンプルが生成される場合は，“サンプルを有する”と判断する. メディア処理 API の API ドキュメントは，Swagger UI で公開しているため，本稿で提案した方法を用いて測定を行った．

(2)尺度「リクエストパラメータサンプル記述網羅率」

Web 上に公開されている API ドキュメントを参照しながら人手で測定した．

(3)尺度「レスポンスプロパティサンプル記述網羅率」

OAS2.0 形式のファイルに記載されたレスポンスプロパティの仕様とそのサンプルを取得して測定した．レスポンスは構造が複雑であることが多く人手では測定困難なためツールを作成した．ツールでの測定手順を図 6 に示す．なお，API プロバイダから OAS2.0 形式のファイルが公開されていない場合は，ファイルを作成した上で測定した．

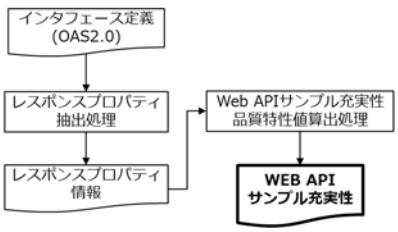

図 6：レスポンスプロパティの測定手順

Web API サンプル充実性の品質評価関数には各尺度の重要度を示す係数が含まれる．我々は，一対比較法を行い各尺度の重要度を決定し，その値を品質評価関数の係数として用いることにした[29].

測定にあたり，前述のサービスが提供する Web API の一部を対象とした. 測定対象の Web API 数を以下の表に示す.

表 5：測定した Web API 数

サービス名	尺度(1)(2)	尺度(3)
Uber	10	4
WordPress	10	10
OpenStack	11	993
メディア処理 API	10	34

各尺度の係数，尺度の測定結果，および品質特性の品質評価値を表 6 に示す.

表 6：測定結果

	尺度			品質特性
	(1)	(2)	(3)	Web API サンプル充実性
係数 / サービス名	0.77	0.16	0.06	－
Uber	1.00	0.82	0.91	0.96
WordPress	0.90	0.02	0.00	0.70
OpenStack	0.00	0.37	0.95	0.12
メディア処理 API	1.00	0.87	0.25	0.92

6.2 安定性の評価

安定性の 2 つの尺度を OpenStack の中心的なサービス群から Nova (Compute Service)，Cinder (Block Storage)，Glance (Image Service) を選択して測定した．測定対象として OpenStack を採用した理由は以下である．

(1) Web API の時系列変化が確認できるデータが取得可能
(2) Web API のインタフェース仕様が機械的に処理可能なフォーマットで管理されている

OpenStack では Web API のインタフェース定義は各サービスのソースリポジトリに格納されている．またフォーマットは Python の reStructuredText を用いており，Web API の仕様を OpenStack 独自のルールに従って記載している[30]．測定対象期間は本フォーマットでのインタフェース定義の管理が始まった 2016 年夏頃(サービスによって若干異なる)とする．

図 7 に測定手順を示す．まず，サービスのリポジトリから各月のスナップショットとなるソースコードを取り出す. 次に Web API のインタフェース定義である reStructuredText (.inc)ファイルを利用し，Web API の仕様記述として普及している OAS2.0 の形式に変換する．OAS2.0 に変換する理由は，Web API 差分を抽出するツールなどの周辺ツールが揃っており，将来様々な Web API が OAS2.0 形式のインタフェースを公開した際に本方法が適用できるようにするためである．

そして，安定性の尺度である，非互換差分量，互換差分量を算出するために各月に発生する Web API 差分を抽出す

る．差分抽出には OSS である swagger-diff[31]を参考に，本品質特性計算用にツールを開発し，適用した．

表 7 に最新月における重み付き非互換差分量[1]，重み付き互換差分量 [1]，安定性を示す．5.4.1 で述べたように非互換の変更は API コンシューマに大きな影響を与えるため安定性への係数に反映する必要がある．そこで，習得容易性と同様に一対比較法[29]を行い，各尺度の重要性を決定し，互換差分量を 1 と捉えた場合に非互換差分量を 10(とても重要)と捉え，互換差分量係数:非互換差分量係数=1:10 となるように互換差分量係数 k_c=1/11，非互換差分量係数 k_i=10/11 とした．測定結果より Cinder は重み付き非互換差分量が多く安定性が低いことがわかる．逆に Glance は重み付き非互換差分量互換差分量，重み付き互換差分量ともに少なく，安定性が高いことがわかる．

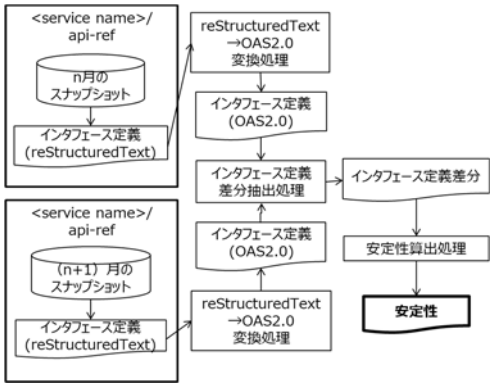

図 7　安定性の測定手順

表 7　安定性の測定結果

サービス名	重み付き非互換差分量 [1]	重み付き互換差分量 [1]	安定性[2]
Nova	1.50	0.06	0.39
Cinder	11.46	2.51	0.07
Glance	0.00	0.02	0.98

7. 考察

7.1 RQ1: Web API の品質モデルは何か?

我々は，Web API の品質特性として個別に提案されてきた Web API のユーザビリティなどの品質特性[11][12][13]の概念を整理し，製品品質標準 ISO 25030[23]を拡張して分類して，Web API の品質特性を提案した．API コンシューマの視点を中心に整理したが，システム API の品質特性との差を明確化する議論がさらに必要である．

品質特性に反映すべき，システム API と異なる Web API の特徴として，“インタフェース定義”と“独立した進化”を指摘した．それに着目して，Web API を用いたアプリケーション開発の初期に API コンシューマにとって重要となる習得容易性と安定性を定義した．

習得容易性は，API ユーザビリティの一特性である．Web API の“インタフェース定義”が REST の原則に従うリソース表現として XML や JSON 等で定義され，ソフトウェアの仕様記述と類似の性質を有することを品質モデルに反映した．

安定性は，システム API と異なり実行時に API ユーザとは独立に進化する Web API の特徴に着目して，Web API の構成要素の変更量に基づいて品質モデルを定義した．

なお，今後も継続する品質特性の議論のために品質モデルのメタモデルを最初に定義した．この考え方は，GQM [32]による品質特性から尺度を導き出すアプローチの一実現手段ともいえる．

7.2 RQ2：提案する品質特性の尺度とその測定方法は何か?

習得容易性に関して 3 つの尺度を定義し，その中で達成度合いを機械的な評価に適していると考えられる Web API サンプル充実性の尺度を定義し，実測可能であることを示した．安定性では，互換性に着目した尺度と測定方法を定義し，実測可能であることを確認した．

7.3 RQ3：提案方法は実際の Web API に有効か?

7.3.1 習得容易性

品質評価関数の有効性を確認するため，本稿の著者を含む 7 名で API コンシューマを被験者としてサービスの習得容易性を評価する実証実験を実施した．全ての被験者は，評価対象とした 4 サービスの Web API を用いたアプリケーション開発の経験はない．

表 8：習得容易性の評価結果の比較

サービス名	習得容易性(Web API サンプル充実性)の評価結果	実証実験での評価結果
Uber	0.94 <1>	0.9 <1>
WordPress	0.55 <3>	0.3 <3>
OpenStack	0.21 <4>	0.2 <4>
メディア処理 API	0.89 <2>	0.8 <2>

被験者は API ドキュメントを参照した開発の容易性を 3 段階で評価した．表 4 に示したサービスに対し，習得が容易と見なしたものを 1，困難を 0，中間を 0.5 でスコアリングを行い，回答結果の平均値を算出した．それらの結果を表 8 に示す．表中の<番号>は品質評価値の順位を示す．本稿では「習得容易性」のうち「Web API サンプル充実性」に着目した．以後の議論では「Web API サンプル充実性」の品質評価値を「習得容易性」の品質評価値とみなす．

6.1 で算出した習得容易性の品質評価値と実証実験での評価値の順位が一致した．従って，尺度，測定方法，品質評価関数は妥当であると判断できる．また，尺度の測定には API ドキュメントのみを用いたので，開発初期に適用可

1 式(4)に示す，時間と Web API コンシューマへの影響の重みを考慮に入れた差分量

2 Nova は 2018/2/1 時点，Cinder と Glance は 2018/1/1 時点で算出

能となる．

各サービスの品質評価値について考察する．

Uber は全リクエストにサンプルがあり，必須のパラメータやプロパティ以外は 1 個以上のサンプルを記載することが高評価につながった．

OpenStack の品質評価値が低いのは，リクエストのパラメータが階層構造を持っている Web API だけにしかサンプルが記載されていないからある．OpenStack の API コンシューマは OpenStack の API プロバイダでもあり，現状は問題視されていないと推察している．しかし，一般の API コンシューマにとっては習得が困難であると判断した．

WordPress はリクエストのサンプルはあるものの，オプショナルなパラメータに全くサンプルを提示しない．さらにレスポンスのプロパティはサンプルのみならず定義も全く記載しないため品質評価値が下がる．

メディア処理 API は，API ドキュメントとしての記述項目の充実を図るための指導を受けていた．その指導によりリクエスト側の記述が非常に充実し，高評価につながった．一方，レスポンス側のサンプルが不十分であることが明確になったので，API プロバイダに対して改善に向けた提言を行うことができた．

Uddin らは[14]で API ドキュメントの問題を 10 個挙げ，それらを発生頻度，影響の大きさで分類している．その中で深刻さが高い問題として不完全さ(Incompleteness)，曖昧さ(Ambiguity)，説明不足(Unexplained examples)，不正確さ(Incorrectness)の 4 つを挙げている．これらの中で，曖昧さと説明不足の問題は，ドキュメント中のサンプルが軽減できることが実証されている．これら二つの問題を，本稿で示した習得容易性によって定量的に評価できるようになる．なお，不完全さは他の Web API とのインタラクションの説明不足の問題，不正確さは API ドキュメントと実際の動作に差異があることによる問題である．不完全さは複数の API のドキュメントの横断的な分析が必要であり，不正確さの評価は Web API の実際の動作を確認する必要があるため，本稿で示した習得容易性では定量化できない．

7.3.2 安定性

OpenStack の公式な情報には安定性を示すものはないが，成熟度を示す Maturity が公開されている[33]．これはサービス利用の判断に使う目的で，自動または手動でつけられたタグをベースに 7 段階の値で表現されており，安定性(Stability)と持続可能性(Sustainability)を示している．この特性は本稿で提案した品質特性の安定性と類似しているため，Maturity と安定性の値を比較することで，安定性の妥当性について考察する．

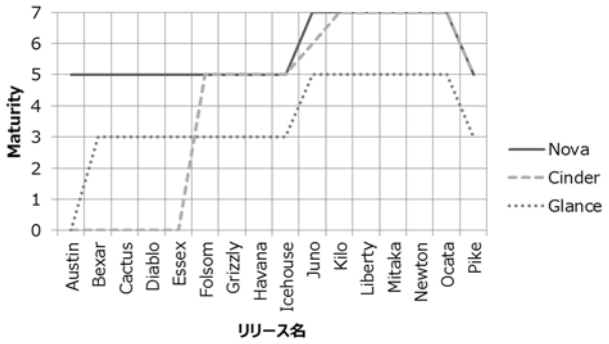

図 8 公式サイト提供のリリースごとの Maturity

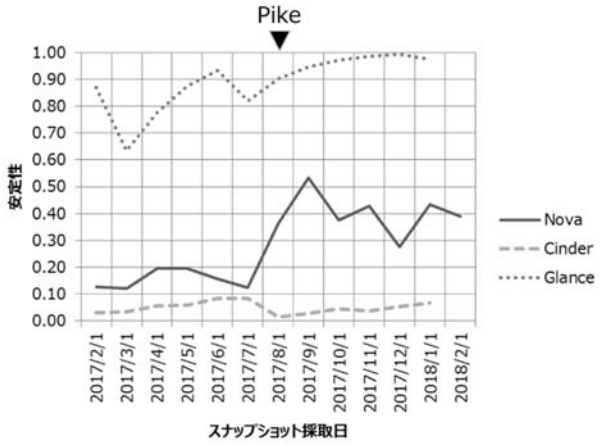

図 9 安定性の月ごとの推移

図 8 に Nova，Cinder，Glance の Maturity をリリースごとにプロットした．各サービス間の Maturity を比較すると，Nova＞Cinder＞Glance の順となっている．安定性については表 7 に示した通り Glance＞Nova＞Cinder の順に高くなっている．Nova＞Cinder については同じ傾向を示しているが，Glance は，Maturity は低いが安定性は高いという結果となった．これは Maturity が安定性だけでなく持続可能性も評価しているのに対して，安定性はインタフェース定義の変更量のみを評価しているためと推測される．

また，リリースを重ねるごとに Maturity は上昇していく．そこで，安定性にも同じ傾向があるかを各月における安定性を算出して確認した(図 9). 安定性に関しても上昇傾向が見られる．ただし，Nova，Cinder，Glance ともに 2017 年 7 月の周辺で一時的に安定性が低下している．これは 2017 年 8 月 30 日の Pike のリリースと関係があると推測できる．図 8 は Pike のリリース後に公式サイトで確認した Maturity であるが Pike で低下していることが確認できた．他のリリースと比較してリリースからの経過日数が少ないことに起因すると推測されるが，安定性においても同様な傾向が確認できた．

上記の議論から，公式サイトが提供する安定性を示す Maturity と品質特性として定義した安定性は類似の傾向を示すと言える．この結果から安定性の品質モデルは，Web

APIの変更の評価尺度として妥当であり,開発初期でのAPIコンシューマにとって有用な情報を提供できるといえる.

8. 今後の課題

本稿の測定では尺度の計算で利用する重みについて，重み設定時の条件を満たす値を業務経験に照らし合わせて選択した．今後さまざまなドメインでの測定を通して適切な値を見つけ出す必要がある．また，本稿で定義した習得容易性と安定性は，アプリケーション開発の初期を想定してAPI ドキュメントを解析対象とした．Web API のインタフェース定義と実際の動作に齟齬がある実情があり，テストや動作確認を伴う表 1 の「API 内容」に関する品質特性の議論が今後必要である．

9. まとめ

クラウドの普及に伴い様々なサービスをWeb APIとして提供し，アプリケーション開発に活用するようになっている．我々は企業システム開発における Web API の利用を想定して，Web API の品質モデルを提案した．アプリケーション開発に関わるステークホルダを整理した上で，API コンシューマの観点から重要な品質特性として，API ユーザビリティの習得容易性と変更の安定性の概念に着目し、習得容易性と安定性の品質モデル，尺度，測定方法を提案した．提案した品質特性の測定方法を実際の Web API に適用して，測定可能であること，及び，人による評価やサービスの成熟度の公開情報と整合性を持つことを示した．このことから，提案した品質モデルはアプリケーション開発の初期段階でWeb APIの品質評価に有効であることを確認した．

産業界では既にWeb APIを用いたアプリケーション開発が増加している．今後，Web API を利用したアプリケーション開発の諸問題を解決するための研究を拡充していく．

参考文献

[1] Basole, R. C., Accelerating Digital Transformation: Visual Insights from the API Ecosystem, IEEE IT Professional, Vol. 18, No. 6, Nov.-Dec. 2016, pp. 20-25.
[2] Iyer, B. and Subramaniam, M., The Strategic Value of APIs, Harvard Business Review, Jan. 2015, https://hbr.org/2015/01/the-strategic-value-of-apis.
[3] Fielding, R. T., Taylor, R. N., Erenkrantz, J. R., Gorlick, M. M., Whitehead, J., and Khare, R., Reflections on the REST Architectural Style and “Principled Design of the Modern Web Architecture”, Proc. of ESEC/FSE 2017, ACM, Sep. 2017, pp. 4-14.
[4] ProgrammableWeb, https://www.programmableweb.com/
[5] 岡部 一詩, API 経済圏, 日経コンピュータ, 2016/5/26 号, pp. 18-31.
[6] De, B., API Management: An Architect's Guide to Developing and Managing APIs for Your Organization, APRESS, 2017.
[7] Espinha, T., Zaidman, A., and Grosset, H., Web API Growing Pains: Loosely Coupled Yet Strongly Tied, J. of Systems and Software, Vol. 100, Feb. 2015, pp. 27-43.
[8] Romano, D. and Pinzger, M., Analyzing the Evolution of Web Services Using Fine-Grained Changes, Proc. of ICWS 2012, IEEE, Jun. 2012, pp. 392-399.
[9] Wang, S., Keivanloo, I. and ZouS Y., How Do Developers React to Restful API Evolution?, Proc. of ICSOC 2014, LNCS Vol. 8831, Springer, Nov. 2014, pp. 245-259.
[10] Wittern, E., Ying, A., Zheng, Y., Laredo, J. A., Dolby, J., Young, C. C., and Slominski, A. A., Opportunities in Software Engineering Research for Web API Consumption, Proc. of WAPI 2017/ICSE 2017, IEEE, May 2017, pp. 7-10.
[11] Menascé, D. A., QoS Issues in Web Services, IEEE Internet, Vol. 6, No. 6, Nov.-Dec. 2002, pp. 72-75.
[12] McLellan, S. G., Roesler, A. W., Tempest, J. T. and SpinuzziLellan, C. I., et al., Building More Usable APIs, IEEE Software, Vol. 15, No. 3, May/Jun. 1998, pp. 78-86.
[13] Myers, B. A. and Stylos, J., Improving API Usability, CACM, Vol. 59, No. 6, Jun. 2016, pp. 62-69.
[14] Uddin, G. and Robillard, M. P., How API Documentation Fails, IEEE Software, Vol. 32, No. 4, Jul.-Aug. 2015, pp. 68-75.
[15] Sohan, S. M., Maurer, F., Anslow, C. and Robillard, M. P., A Study of the Effectiveness of Usage Examples in REST API Documentation, Proc. of IEEE VL/HCC 2017, Nov. 2017, pp. 53-61.
[16] Stylos, J., Faulring, A., Yang, Z. and Myers, B. A., Improving API Documentation Using API Usage Information, Proc. of VL/HCC 2009, IEEE, Sep. 2009, pp. 119-126.
[17] Robillard, M. P., What Makes APIs Hard to Learn? Answers from Developers, IEEE Software, Vol. 26, No. 6, Nov./Dec. 2009, pp. 29-34.
[18] Murphy, L., Alliyu, T., Macvean, A., Kery, M. B. and Myers, B. A., Preliminary Analysis of REST API Style Guidelines, Proc. of PLATEAU 2017/SPLASH 2017, ACM, Oct. 2017, pp. 1-9, https://2017.splashcon.org/track/plateau-2017.
[19] Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E. and Lau, A., An Empirical Study on Web Service Evolution, Proc. of ICWS 2011, IEEE, Jul. 2011, pp. 49-56.
[20] Li, J., Xiong, Y., Liu, X. and Zhang, L., How Does Web Service API Evolution Affect Clients?, Proc. of ICWS 2013, IEEE, Jun.-Jul. 2013, pp. 300-307.
[21] ISO/IEC 29148, Software and Systems Engineering- Life Cycle Process - Requirements Engineering, 2011.
[22] ISO/IEC 25010:2011, Systems and software engineering —Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models, 2011.
[23] ISO/IEC 25030:2007, Software Engineering- Software Product Quality Requirements and Evaluation (SQuaRE) - Quality Requirements, 2007.
[24] ISO/IEC 15939:2017, Systems and Software Engineering - Measurement Process, 2017.
[25] Raemaekers, S., Deursen, A. v. and Visser, J., Measuring Software Library Stability through Historical Version Analysis, Proc. of ICSM 2012, IEEE, Sep. 2012, pp. 378-387.
[26] Bug Prediction at Google, http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html.
[27] Swagger UI, https://swagger.io/swagger-ui/
[28] Open API Specification(version 2.0), https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#schema
[29] Mu, E. and Pereyra-Rojas, M., Practical Decision Making using Super Decisions v3, Springer, 2017.
[30] OpenStack API Documentation, https://docs.openstack.org/doc-contrib-guide/api-guides.html.
[31] Swagger-diff, https://github.com/civisanalytics/swagger-diff.
[32] Basili, V. R. and Rombach, H. D., The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Tran. on Software Engineering, Vol. 14, No. 6, Jun. 1988, pp. 758-773.
[33] OpenStack Project Navigator, https://www.openstack.org/software/project-navigator/