An Approach for a Fast Cost Validation of Web-Based APIs supported by Functional Size Measurement with COSMIC

Sandro Hartenstein[1,2], Konrad Nadobny[1,2],
Steven Schmidt[1,2] and Andreas Schmietendorf[1,2]

[1] Berlin School of Economics and Law, Alt-Friedrichsfelde 60, 10315 Berlin, Germany
[2] Otto-von-Guericke-University Magdeburg, Universitätspl. 2, 39106 Magdeburg, Germany

andreas.schmietendorf@hwr-berlin.de
schmiete@ivs.cs.uni-magdeburg.de

Abstract. Web-based APIs are increasingly important for software engineering. Selecting a respective API and comparing the cost and value of different APIs is currently very difficult due to the diversity of used price models and provided functionality. In order to improve the situation, we propose the use of an independent description of the functional size of an API that allows a fast cost validation based on the provided functionality. Our approach makes use of the COSMIC method and applies it to the functional specification of a Web API. Following this approach, we are analyzing the Swagger (openAPI) specification and derive COSMIC FP from it. The approach is described under consideration of a real Web API from the Telecommunication industry. The derived COSMIC FP can be helpful for empirical analyses of several cost drivers, as described in the paper too.

Keywords:
API Economy, Web API, API-Specification, OpenAPI, Swagger, COSMIC

1 Motivation and Background

The ability to develop software fast, efficient and collaboratively is a core success factor to cope with the challenges of digitalization. The industry demands shorter development cycles, faster delivery times, well-integrated holistic solutions and all of this under enormous cost pressure. In this environment software engineering needs to be highly efficient and effective. Web-based APIs (Applications Programmers Interface – further APIs) are a core element to deliver software more efficiently. Developers can integrate existing services easily using APIs. Ideally these APIs are Open APIs, which means that they follow a certain specification standard set by the Open API Initiative and are publically available.[1] The services exposed, consumed and integrated via APIs can be as simple as raw sensor data, more complicated business data and system functionality or even whole business processes. APIs enable the easy integration and service orchestration inside the business borders and far beyond. They are the basis for Service-orientated Architectures, applications driven by Micro Services as well as mobile Applications (Apps). APIs enable developers to create powerful software in an agile and efficient matter by integrating already existent third-party functionality into their product. This makes APIs an enabler and the backbone of digitalization.

There is an emerging market for APIs. In this market API Providers offer services that are being used by API Consumers.[2] Some big API Providers like Twitter, Amazon or Google do create more revenue through their APIs than through their user-centric applications. *“The term API Economy was shaped in the context of monetarization and trading of APIs and can be defined as the economic actions that are based upon the automated, multilateral, dynamic and relatively anonymous competence network of highly specialized partners.”* [3] Within this market there are different pricing models for APIs. Finding a plausible price that works for the API Provider and is accepted by the API Consumer is important for the success of an API.[4]

This paper elaborates on the question whether the COSMIC-Method can be used to create measure the functional extend of an API and determine plausibility of API pricing. Does it allow for comparability with similar offers on the API market? Is it a suitable method instrument to estimate resulting consumer-sided costs of using the respective API?

The first part the paper provides an analytical overview on possible API pricing models for APIs. The second part identifies the cost-driving elements in the API lifecycle both from the API Provider and the API Consumers perspective. After this, the paper focusses on the API Consumer perspective and applies the suggested COSMIC-based concept. The COSMIC-FP methodology is being used to determine the functionality of a provided API. This API is being determined exclusively via the provided Swagger / Open API Specification. Details about the implementation are hidden to the API Consumer. Alternative approaches like WSLD or API Blueprint are not in scope for this. In the last part the paper discusses the possibility to extend the methodology to non-functional requirements. The concept is validated by applying it to a Swagger-specified example API.

2 Current pricing models

In order to validate the costs of different API, one needs to take a look at the different pricing models. In general there are four different pricing model groups: Free, API Consumers Pay, API Consumers get payed and Indirect pricing. [5]

About the half of all public APIs are free to use.[6] There is no limit regarding the timeframe, the quota or any other form of usage.[5] The API Provider covers the whole costs without requesting a payment from the API Consumer.[7]

When API Consumers pay, there are different ways, how this can happen. Usually the API Consumer pays a fee to use the service based on the pricing model.[5] Pay-per-use is the least complex pricing model and charges the per unit cost.[5] This can be counted by object or by simply calling the incoming API calls.[8] This can also be an transaction fee, that can be applied as a percentage of fixed price per transaction.[8] Similar approaches are a Unit-based approach with pre-payed access quotas[6] or a Subscription model with a limited validity of time[8]. Tiered subscription with different usage levels, allow the API Consumer to use the API to certain extend based on their payment.[5] Closely connected to this the Freemium pricing model, where a basic service is provided for free, while more professional users need to pay. The Overage pricing model can be used where a pay-per-use fee applies to every API call beyond the quota limit of the package until the next package is activated.[6] About a quarter of all public APIs are using some kind of Freemium approach.[6]

In addition to these monetary models, there is also the Exchange-based payment: retrieval of the data itself does not require payment but it is expected to receive processed information using this data back from the recipient to make use of that dataset and monetarize it somewhere else.[7]

When API Consumers get payed, they are usually be rewarded for making use of the API. These APIs are offered on a free-to-use basis by the API Providers who share their profits with the API Consumers based on the respective model.[5] This can be a Share of the Revenue generated via the API, or an affiliate model with commissions for user-interactions via the API.[5] This model might be only available from a certain exposure.[6]

Indirect pricing models do not rely on a direct monetarized exchange, but passes through the monetarization realized indirectly through the use of the API. SaaS-Solutions for example often come with APIs in a package with the other Services.[5] The API can also collect different generated data from usage and relies on transactions through the APIs service, e.g. for marketplaces.[9]

Currently there is no unique approach to describe the functional and non-functional behaviors of a Web APIs in a generic manner. Each Web API offering provides an own procedure to describe her quantitative and qualitative behavior.

3 Cost-drivers in the API Lifecycle

When it comes to the costs linked to an API, taking a holistic approach is helpful. As described in the previous section (Current Pricing Models), there are API Providers and API Consumers.[2] While the API Provider is being compensated for his efforts based on the respective pricing model described in the previous section, the costs for the API consumer require a closer look.[10]

While in the API economy everyone could be both API Provider and API Consumer at the same time, in order to determine the isolated costs of one particular API it is important to distinguish between the costs of providing and the costs of consuming a respective API. The Full Lifecycle API Management Model by CA Technologies[3] gives a good overview about the required efforts. The API Consumer undertakes efforts to discover, develop, consume, manage and monitor the API.

The better the design and specification of the API, the lower the costs for the API Consumer. API marketplaces like Google Cloud APIs, Programmable Web, or Rapid API are easily searchable, provide an API catalogue and classification, support the registration and authorization and are an enormous help for developers to discover APIs effectively and efficiently.[11] The effort for using an API for development is highly dependent on the functional design of the API itself on the one hand and the quality of the specification provided by the API Provider on the other hand.[12]

Once integrated to the API Consumers application the API is being called whenever needed. This is where the pricing model becomes relevant.[2] Also costs for API Management and Monitoring need to be considered here. API Consumers also need to assess the costs of getting up to speed with the usage of the API, how well it fits into their API environment and infrastructure and in addition take a closer look at potential cost-intensive risks regarding non-existing Service Level Agreements (SLAs) or possible low service quality of the APIs.[12] A good API management is critical to cope with the agile reality of API evolution. Not only does it allow keeping track of changes to respective APIs, but it also ensures a continuous development of the API-driven competence network.[13]

The ideal API would have a Price-Model fitting well to the expected operational usage and would be easy to connect to and easy to use. Following well-established industry Standards like Open API, REST and OAuth2.0 can help to keep the additional costs to a minimum.

4 COSMIC FP related strategy and mapping approach

In order to estimate the costs linked to an API, the COSMIC cost estimation method is being linked with the API specification framework Swagger. While COSMIC provides a measuring model for the functional size of the API, Swagger provides a functional specification of all methods and attributes exposed by the API Provider. Due to the frequent use of Swagger in the industry, it is being used as an example for a standardized machine-readable functional specification of API functionality.

Swagger is an open source framework for creating, designing, documenting, and using RESTful Web services, closely connected to Open API: *“It offers a set of rules to semantically describe an API like a blueprint for a house.”* [14]

The structured data formats (mostly in JSON – Java Script Object Notation) can be easily read by machines and people. The API specifications can be established as contract first or as code first approach (than with annotation in the source code). Over it out the Swagger approach provide an abstraction from technology and implementation details. Furthermore the implementation details of a web based API are hidden by the Swagger specified interfaces.[15]

From the author’s point of view, Swagger can therefore be used to derive the functional user requirements and on this basis the functional size of an API offering. While the functional size is a cost driver on the API Providers site, the measurement and comparison of functional size is immanent for the API Consumer to compare the complexity of different APIs and the respective costs models. This is why this paper focusses on the API Consumers point of view.

The goal is to establish a generic measure (functional size oriented) to support the following development oriented tasks: Selection of Web API offerings, Compare different Web API offerings, Comprehensible of the function scope, Estimation of integration efforts, Estimation of test related efforts, Estimation of operating efforts, Collection of empirical experiences.

The measure should be based on the functional size of an API-offering that can be determined based on the functional specification. This specification should be machine-readable (like Swagger) to enable an automated measurement. In this context it is important to understand the concept of Swagger. A Swagger file provides the following information in a JSON-based format[15]:

- **METADATA.** Swagger version, title and further documentation
- **SERVERS.** Definition of one or more Servers and the corresponding URLs
- **PATHS.** Provided Endpoints and HTTP based operations
- **PARAMETERS.** Data types for query strings, headers or cookies
- **REQUEST BODY.** Description of body content and media types
- **RESPONSES.** Status code like 200 ok or 404 not found
- **INPUT and OUTPUT MODELS.** Use of common data structures
- **AUTHENTICATION.** methods like API key or OAuth 2

In the context of counting, one needs to consider especially the PATH **(**Functional user requirements = FUR), PARAMETER (as data entries = symbol E) and RESPONSES (as data exits = symbol X). The INPUT and OUTPUT MODELS are features of the latest Swagger 3.0 specification and therefore rarely used in the industry. This is why these sections are not part of the current model, even though they might be taken into account at a later stage of research, as a model-based description would enable a more exact counting of the data movements.

Table 1. COSMIC mapping approach

HTTP Operation COSMIC FUR	**CRUD paradigm**	**Possible Mapping COSMIC FP**	**Implicit Mapping**
GET resource	Read	Exit ($X_{response}$) (+ Entry ($E_{request}$))	Read (Rimplicite)
GET resource list	Read	Exit ($X_{response}$) (+ Entry ($E_{request}$))	Reads (nR)
POST resource	Create	Entry ($E_{contents}$) (+Exit (X_{commit}))	Write (Wimplicit)
POST on TASK resource	Create	Local extension for TASK	-
PUT resource	Update	Entry ($E_{contents}$) (+Exit ($X_{response}$))	Read and Write (R+W)
DELETE resource	Delete	Exit ($X_{response}$) (+ Entry ($E_{request}$))	Write (Wimplicit)

Swagger supports the HTTP based operations Get, Post, Put, Patch, Delete, Head, Options and Trace. As shown in Table 1 these can be mapped to the CRUD paradigm and data movements, while the relevant HTTP functions are only Get, Post, Put and Delete. Going forward, the other HTTP operations will be ignored in this context.

Following the COSMIC approach[16], functionality of software is described by four data movement (Entry, Exit, Read, Write) types. The measurement unit is 1 CFP (Cosmic Function Point) per data movement. The size of a functional process within software is then the arithmetic sum of its data movement types.[16]

$$\text{Size}(\text{functional}_{\text{process}}) = \Sigma\,\text{Entries} + \Sigma\,\text{Exits} + \Sigma\,\text{Reads}_{\text{implicit}} + \Sigma\,\text{Writes}_{\text{implicit}}$$

The mapping possibility to hidden data movements behind the question mark can be characterized as implicit. This means a HTTP-Get operation deals for example with selection parameters (Entry) and with reading from the persistence layer (Read). Therefore we are able to assume this kind of data movements. For the API Consumer, it is unknown, which data movements are happening behind the API facade and additional software layers for example covering up potential legacy systems are hidden behind the API Boundary.

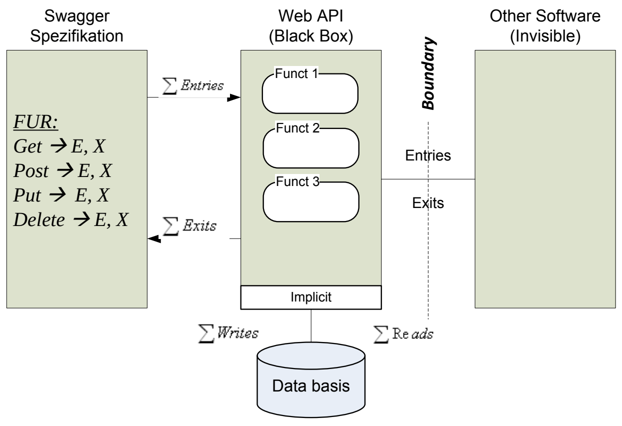


Figure 1. Concept of the counting with COSMIC

As a result of the concept of APIs, the full extent of data movements in the software is not visible to the API Consumer. This is why only the entries and exits defined in the available Swagger specification are being count. This results in a measurement of the functional size of the interface specification, not the whole layered service architecture. However, as the complexity of the underlying architecture is not relevant for the API Consumer, this measurement is well-suited for measuring the functional size of an API.

5 COSMIC FP related measurement example

The following section contains an example of how the COSMIC-FP method can be applied to the measurement of the functional size of an API based on a Swagger specification file. The concept has been applied as a prototype using the TMF Location Service API as an example. The scope of the measurement considers the functional user requirements as defined inside the Swagger based specification for the TMF

Location Service API. The API provides a description of the geographically correct address of an object based on a textual input provided by the API Consumer. This description can be a Coordinate, a point, a line or an area.[17]

The corresponding Swagger-file has a size of 1421 LoC (Lines of Code) and describes four different functional processes, as listed in Table 2. The HTTPS based operations deals with Get (data receive), Post (data send) and Delete (data delete). An extract of the Swagger file is shown in Figure 3 on the next page.

After identifying the functional processes, the data movements of each functional process are being analyzed. In order to determine the functional size, the respective data movements are being count and liked to the previously identified functional process, as shown in Table 2.

Table 2. Example of Counting CFP based on the operations specified in the Swagger File

	Data groups	Entries E	Exits X	Reads R	Writes W	Sum CFP
GeographicLocation (GL)						
GET List GL entities	3	3	1	1	0	**5**
GET Retrieve a GL entity	2	2	1	1	0	**4**
RetrieveGeographicLocation (RGL)						
POST Create a RLG entity.	1	1	1	0	1	**3**
GET List RGL entities.	3	3	1	1	0	**5**
GET Retrieve a RGL entity.	2	2	1	1	0	**4**
RetrieveLocationRelation (RLR)						
POST Create a RLR entity	1	1	1	0	1	**3**
GET List RLR entities	3	3	1	1	0	**5**
GET Retrieve a RLR entity	2	2	1	1	0	**4**
Hub						
POST Register a listener	1	1	1	0	1	**3**
DELETE Unregister a listener	1	1	1	0	1	**3**
Total COSMIC Function Points:						**39**

In addition to the functional endpoints, the provided Swagger-specification-file contains a description of the class model. This can be used to analyze the corresponding data groups, as described in a UML class models or defined in the Swagger file. Figure 2 shows the class model for the RetrieveGeographicLocation with four data groups used differently throughout the functional processes.

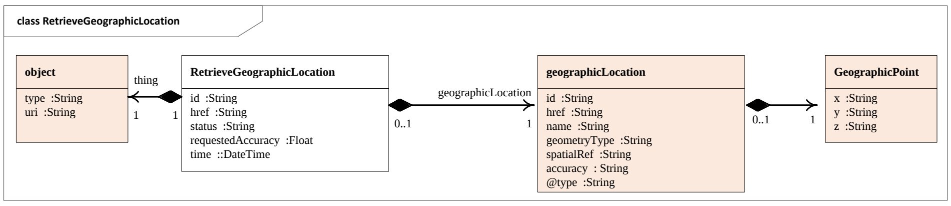
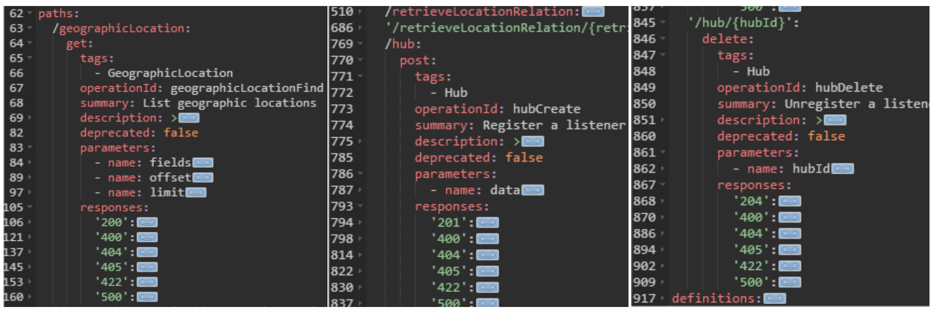


Figure 2. UML-class model[18]

A specific mapping problem can be found with GET. The response can be a single entity, as well as a list of entities. From the author's point of view, it could be useful to take another weighting in the case of a retrieved list (n exits). With the existent approach however, the analysis results in a number of 39 Cosmic Function Points. Dumke et al. provide two conversation Keys to anticipate the efforts linked to the functional size measured with CFP:

1CFP ≈ 0,07 PM (for SOA Architectures) [19 p.399] and 1 CFP = 40.8 LoC [19 p.452]

Following this anticipation, the measured functional size of 39 CFP corresponds to an effort of 2.45 Person Months (PM), or an application size of 1591 LoC.



```
paths:
  /geographicLocation:
    get:
      tags:
        - GeographicLocation
      operationId: geographicLocationFind
      summary: List geographic locations
      description: >
      deprecated: false
      parameters:
        - name: fields
        - name: offset
        - name: limit
      responses:
        '200':
        '400':
        '404':
        '405':
        '422':
        '500':
```

```
  /retrieveLocationRelation:
  '/retrieveLocationRelation/{retr
  /hub:
    post:
      tags:
        - Hub
      operationId: hubCreate
      summary: Register a listener
      description: >
      deprecated: false
      parameters:
        - name: data
      responses:
        '201':
        '400':
        '404':
        '405':
        '422':
        '500':
```

```
  '/hub/{hubId}':
    delete:
      tags:
        - Hub
      operationId: hubDelete
      summary: Unregister a listen
      description: >
      deprecated: false
      parameters:
        - name: hubId
      responses:
        '204':
        '400':
        '404':
        '405':
        '422':
        '500':
definitions:
```

Figure 3. Example of the TMF Location Service API Swagger File [18]

6 Summary

This paper has shown a first approach to count the functional size of a web-based API under consideration of the Swagger specification. The research is not yet finished and several further activities are planned, like: Discussion of the approach inside the COSMIC community, Validation of the counting correctness, Implementation of a Swagger parser for an automatic counting, Widening of the empirical experience (application for a high amount of APIs), Deposit of the COSMIC FP inline the Swagger specification, Discussion with API marketplace providers.

Even though the prototype shows that the COSMIC-FP methodology is suitable in the context of API measurement, some weaknesses to the approach where identified. Especially the mappings between the HTTP operations and the data movements are weak. It is questionable, whether mapping each HTTP-operation to a functional process is correct. Further research will generate valuable insights about this aspect. Another aspect deals with the handling of the data groups, our counting approach has considered each as an own data movement. Overall, it could be useful to identify data attributes to reach better details. Potential efforts are not exclusively linked to functional user requirements (FUR), as quality user requirements (QUR), platform user requirements (PUR) and process or organizational requirements (POR) also might be considered as relevant.[20]

References

1. Open API Initiative Homepage, www.openapis.org, last accessed 7/22/2019
2. Chiu, D.: Where API management is headed in 2017, CA technologies (2017)
3. Resch, O.: API-Economy − eine Situationsbestimmung,
   in Tagungsband BSOA/BCloud2015, Shaker-Verlag Aachen (2015)
4. F. Stahl, A. Löser, G. Vossen: Preismodelle für Datenmarktplätze.
   Informatik Spektrum 38, pages 133–141 (2014)
5. Bustos, L.: 18 API Business Models Deconstructed, GetElastic (2013)
6. A. Walling, API Pricing Strategy Webinar, pages 18ff (2017)
7. J. Musser, Open APIs: What's Hot, What's Not?, pages 39ff (2012)
8. Kirchoff, L.: The Ultimate Guide to Pricing Your API, Nordic APIs (2017)
9. Google AdSense Host API, developers.google.com/adsense/host, last accessed 7/22/2019
10. Masse, N.: Full API lifecycle management: A primer, Red Hat (2019)
11. Wood, C.: The Evolution of the API Marketplace, Nordic APIs (2017)
12. Norman, D.: How to make Swagger codegen work for your Team, CapitalOneTech (2018)
13. Doerrfeld, B. C.; Pedro, B.; Sandoval, K.; Krohn, A.: The API Lifecycle -
    An Agile Process For Managing the Life of an API, Nordic APIs (2015)
14. Costa, J.: How-To: Getting Started with Swagger, Akamai (2019)
15. Swagger Homepage, https://swagger.io, last accessed 7/22/2019
16. ISO/IEC 19761: COSMIC: a functional size measurement method (2017)
17. Geographic Location Management API in TM Forum Open API Table,
    https://projects.tmforum.org/wiki/display/API/Open+API+Table, last accessed 7/22/2019
18. Geographic Location API REST Specification, TM Forum, TMF675, R 17.5.0 p 10 (2018)
19. Dumke, R.; Schmietendorf, A.; Seufert, M.; Wille, C.: The Next Generation of Functional
    Size Measurement – Handbuch der Software-Umfangsmessung und Aufwandschätzung,
    570 Seiten, Logos-Verlag, Berlin (2014)
20. Dumke, R.; Fiegler, A.; Hegewald, H.; Neumann, R.; Wille, C.:
    Established Software Metrics adapting to COSMIC Measurement,
    SoftwareMeasurementNews, Vol. 20 No. 1 (2015)

We would like to thank especially Dr. em. Reiner Dumke for his constructive criticism and especially the discussion of the COSMIC figures have added significantly to the successive maturity of this work.