UTRECHT UNIVERSITY

MASTER'S THESIS

# API-m-FAMM: A Focus Area Maturity Model for API Management

*Author:*
Max MATHIJSSEN
4274873

*Supervisors:*
Dr. Slinger JANSEN
Dr. Gerard WAGENAAR
Michiel OVEREEM

*A thesis submitted in fulfillment of the requirements for the degree of Master of Science*

*in the*

Business Informatics
Department of Information and Computing Sciences

April 26, 2021

*"Before you become too entranced with gorgeous gadgets and mesmerizing video displays, let me remind you that information is not knowledge, knowledge is not wisdom, and wisdom is not foresight. Each grows out of the other, and we need them all. "*

Arthur C. Clarke

UTRECHT UNIVERSITY

# *Abstract*

**API-m-FAMM: A Focus Area Maturity Model for API Management**

by Max MATHIJSSEN
4274873

Organizations are increasingly connecting using Application Programming Interfaces (APIs) to share data, services, functionality, and even complete business processes. However, the creation and management of APIs, is non-trivial. Aspects such as traffic management, community engagement, documentation, and version management are often rushed afterthoughts. In this research, we present and evaluate a Focus Area Maturity Model for API Management (API-m-FAMM), which addresses the domains of Lifecycle Management, Security, Performance, Observability, Community, and Commercial. The API-m-FAMM is a model that organizations may use to assess and evaluate their maturity in API management and to set out a course for systematic improvement and evolution. The model is grounded in literature and practice, and was developed and evaluated through a Systematic Literature Review, 11 expert interviews, and 6 case studies. These evaluations are reported on, and show that the API-m-FAMM is an efficient tool for aiding organizations in gaining a better understanding of their current implementation of API management practices, and provides them with guidance towards higher levels of maturity. Additionally, this study's unique case study design shows that FAMMs can be successfully deployed in practice with minimal involvement of researchers. The Focus Area Maturity Model for API Management is maintained on www.maturitymodels.org, allowing practitioners to benefit from its useful insights.

# *Acknowledgements*

This thesis is the conclusion of my academic education. Even though it was not always easy, in part due to the COVID-19 pandemic and having to write my work largely from home, many people were involved that made this journey much easier.

I would like to thank everyone that has provided me with their feedback, knowledge, and help along the way. Firstly, my gratitude goes out to my academic supervisors Slinger Jansen and Gerard Wagenaar, whose help, time and feedback have been critical in improving and guiding my research. Secondly, a big thank you to my daily supervisor at AFAS, Michiel Overeem. This work would not have been possible without all the invaluable brain storm sessions and discussions we have had this past year, as well as his many proof-reads and excellent feedback. Next, I would like to thank Thymen Simons, Jeroen van Stokken, Machiel de Graaf, and Christian Lankman at AFAS for their participation in the case study, their interest in my work, and providing excellent ideas. Also, thank you to all the experts that have been involved in my research, this would not have been possible without your help and the countless hours of discussions we have had. Lastly, a special thanks to my mom, dad, brother, and friends. I could not have done this without your continuous love, support and understanding!

# Contents

# Chapter 1

# Introduction

In recent years, there has been an increasing demand among organizations to have access to enterprise data through a multitude of digital devices and channels. In order to meet these expectations, enterprises need to open and provide access to their assets in an agile, flexible, secure and scalable manner (De, 2017). These assets include matters such as raw and cleansed data, images, videos, documents, or functionality that performs complex calculations or data processing based on inputs (L. Weir, 2019). Access to these assets may be provided by utilizing Application Programming Interfaces (APIs). An API is a software-to-software interface that defines a contract for applications to communicate with one another over a network, without the need for any user interaction (De, 2017). By adhering to this contract, assets may be exposed to third-parties in a limited fashion, without sharing the code base.

As shown by an analysis conducted by ProgrammableWeb (Santos, 2019), which is the largest directory of APIs, the usage and offering of APIs has evolved from a curiosity to a trend since the year of 2005. This observation is further supported by a survey conducted by Coleman Parkes Research (2017), which shows that 88% of global enterprises have some form of an API program. Furthermore, this survey has found that respondents experience a wide variety of benefits from their API programs, including an average increase in speed-to-market of around 18% (Medjaoui, Wilde, Mitra, & Amundsen, 2018). These statistics signal the emergence of the API Economy, in which organizations are offering access and the ability to recombine their digital services and products for novel value creation (Basole, 2019). As a result, by making their APIs accessible to external or partner consumers, these organizations are able to reach new markets, enable their business strategy and drive the creation of new innovative solutions (Bui, 2018).

However, after an API has been created, it needs to be managed so that developers may easily integrate it into their applications. API management is a discipline that has evolved to deliver the processes and tools required to discover, design, publish, implement, use, operate and maintain APIs (L. Weir, 2019). API management is accomplished by performing activities such as providing helpful documentation, controlling access to the API, as well as monitoring and analysing its usage. Oftentimes, these activities are supported through an integrated API Management platform, which, among other things, helps an organization publish APIs to internal, partner, and external developers to unlock the unique potential of their assets (De, 2017). Organizations may choose from a plethora of commercially available API management platforms, the most prominent of which, according to Forrester Research (2020)'s latest evaluation of API management solutions, include Software AG, IBM API Connect, Google's Apigee, Oracle's WSO2 and Axway's AMPLIFY API Management.

## 1.1 Problem Statement

As highlighted in the previous section of this work, it is evident that for an increasing amount of organizations, the utilization of APIs as part of a successful API program is becoming increasingly important. In fact, it has been found that enterprises with advanced API management processes experience up to 47% better business results than those with basic API management (Coleman Parkes Research, 2017). In an effort to guide organizations in successfully managing these programs, some commercial frameworks and tools exist with which organizations may evaluate and assess their API management approach and capabilities. However, often due to their commercial and industry focused nature, these frameworks are not publicly available, transparent or grounded in academic literature. Despite growing interest in the topic of API Management among the research community, more research is needed in order to fill knowledge gaps and identify best practices regarding the subject from an academic perspective. This is highlighted by the observation that currently, relatively little in-depth literature on API Management exists (Mathijssen, Overeem, & Jansen, 2020a). Additionally, a uniform, comprehensive and widely accepted definition of the topic is lacking within the research community, as well as in the software industry. Furthermore, frameworks or overviews that are grounded in literature and capture all the processes, capabilities and features API Management is comprised are lacking.

Despite many large scale organizations having an API program in place, a significant portion of these are limited in terms of their approach, strategy, features and capabilities. For instance, a survey conducted by Coleman Parkes Research (2017), as mentioned in the previous section, has found that only 51% of respondents were regarded to have an advanced API management approach in place. Furthermore, 37% have what are considered to be basic API management procedures in place, 10% employ only limited capabilities, and the remaining 2% of respondents have no API management capabilities in place at all. However, the respondents of this survey are classified as senior business, and these organization's degree of maturity with regards to API management was assessed using an evaluation framework that only takes a limited amount of capabilities into account. Because of this, it could be argued that a similar survey conducted among smaller organizations, using an evaluation framework that takes a wider range of API management capabilities into account, would yield results that indicate an even lower adoption of API management approaches.

While there are many organizations that successfully manage their API programs, a small amount of these organizations are willing to share their experience and expertise publicly. There are several possible reasons for this reluctance. Firstly, organizations that are performing well in terms of their API management programs might simply not have the time, resources or personnel to share their experience and expertise with third parties (Medjaoui et al., 2018). Secondly, organizations that are apprehensive with regards to the amount of knowledge they share on their API management expertise might consider their know-how to be a competitive advantage, and will as such not feel urged to make their findings public. Finally, even in the event where organizations share their experience at public conferences, articles or blog posts, the information shared is usually company-specific and difficult to translate to a wider range of organizations' API programs (Medjaoui et al., 2018).

## 1.2 Research Objective

As outlined in the problem statement above, several problems and knowledge gaps exist in relation to the topic of API management. Currently, organizations that expose their API(s) to third-party developers have no tools or frameworks with which they may evaluate and improve upon their business processes regarding the topic of API management, that are publicly available and are grounded in both literature and industry, at their disposal. Hence, using the practices, capabilities and features that constitute the topic of API management as encountered in literature as input, the main objective of this work is to:

- Construct, evaluate and validate a framework or tool with which organizations may evaluate, improve upon and assess the degree of maturity their business processes regarding the topic of API management have.

In summarizing this work's objectives, the problem context, artefact, requirements and desired impact of this work is distilled into a goal statement, by using the design problem template as presented by Wieringa (2014).

> **Goal Statement:** This work's goal is to *improve* the transparency and availability of API management assessment frameworks and tools *by* constructing, evaluating and validating a publicly available, industry and academically grounded framework or tool *that can* be used by organizations that expose their API(s) to third-party developers to assess and evaluate their degree of maturity with regards to API management *in order to* improve upon their API management-related business processes.

## 1.3 Background

Serving as a starting point for this thesis, a Systematic Literature Review (SLR) was conducted (Mathijssen et al., 2020a). This SLR was based on the methodology developed by Okoli (2015), as well as guidelines composed by Kitchenham and Charters (2007). First, a comprehensive overview of literature related to API management was collected. This was accomplished by entering a series of relevant keywords in a list of scientific libraries, resulting in the extraction of an initial collection of 5152 books, research papers, theses and white papers. After having applied a set of inclusion and exclusion criteria, as well as removing duplicates, this collection was narrowed down to 43 papers. Next, an overview of all the varying definitions for API Management as encountered in these papers was composed, consisting of 24 unique definitions. Then, as a result of a key-term frequency analysis using these definitions as input, a new and comprehensive definition of the topic was proposed.

Next, the features API Management consists of were identified and extracted in the form of practices and capabilities, which are components of the Focus Area Maturity model, as first introduced by van Steenbergen, Bos, Brinkkemper, van De Weerd, and Bekkers (2010). Maturity models are a proven tool used in the creation of collections of knowledge of practices and processes concerning a particular domain (Becker, Knackstedt, & Pöppelbuß, 2009; Jansen, 2020). One specific type of maturity model is the Focus Area Maturity model (FAMM) (van Steenbergen et al.,

2010; van Steenbergen, Bos, Brinkkemper, van de Weerd, & Bekkers, 2013), which is used to establish the maturity levels of an organization in a specific functional domain. This functional domain is described by the set of focus areas that constitute it (Jansen, 2020). Each focus area is composed out of a set of capabilities. These capabilities are positioned against each other in a maturity matrix. In the scope of this research, capabilities were defined as *the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices*. As a result of scanning and coding the body of included literature, 39 capabilities were identified and extracted. Furthermore, the practices that capabilities are composed of were defined as *any practice that has the express goal to improve, encourage and manage the usage of APIs*. Among the 32 papers that were found to contain at least one practice or capability, 114 practices were identified and extracted. Based on the positioning of the capabilities and practices in the maturity matrix, a number of maturity levels may then be distinguished (Jansen, 2020). These maturity levels may then be used to guide an organization in the incremental development of the functional domain.

## 1.4 Research Questions

Based on the problem statement and research objectives, the following main research question (MRQ) and 5 sub-research questions are formulated, ensuring this research succeeds in achieving its aims, goals and objectives.

*MRQ: How can organizations that expose their APIs to third parties evaluate their API management practices?*

As mentioned earlier, organizations that employ API management activities have no tools or frameworks with which they may evaluate and improve upon their business processes regarding the topic of API management, that are publicly available, transparent, and grounded in both literature and industry, at their disposal.

> *SQ1: What type of tool or framework is best suited for organizations wanting to assess their API management related business processes?*
>
> As part of the process of answering this work's main research question, an artefact type that may be adapted to a framework or tool which aligns with its intended use, as well as being easy to use and understand for its users, should be decided upon.
>
> *SQ2: What elements should the API management assessment tool or framework be comprised of?*
>
> In order to create an API management assessment framework or tool that adequately fulfills it's intended purpose, a complete collection of all the (best) practices, capabilities, features and processes API management is comprised of must be identified.
>
> *SQ3: What are suitable criteria, benchmarks, and methods for evaluating the API management assessment tool or framework?*
>
> In order to improve the effectiveness and value of the framework or tool, a set of criteria, measures and benchmarks needs to be composed and adhered to. Doing so will enable comparison with existing frameworks, tools and artefacts, resulting in an optimal API management assessment framework or tool.

Furthermore, a suitable evaluation method needs to be employed in order to evaluate whether the contents of the tool or framework are complete.

*SQ4: What is a suitable method for validating the API management assessment tool or framework in practice?*

In order to satisfy the requirement regarding the tool or framework being grounded in industry, the artefact must be applied, tested and validated in practice.

## 1.5 Thesis Outline

This thesis is structured as follows. This first chapter comprises the overall scope of this project, a problem statement and the main objectives, relevant background with regards to this study, as well as the research questions that guide this work. Chapter 2 elaborates upon the research design and approach this study adheres to, and includes a description of the various research methods used. Chapter 3 describes all relevant literature in the scope of this research, as well as summarizing and evaluating all existing frameworks, tools, models, reports and case studies that are related to the scope and goal of this study. In Chapter 4, the process of designing the API management assessment framework is traced and described. Next, Chapter 5 describes the results of the expert interviews that were conducted to evaluate the framework through two evaluation cycles. Chapter 6 details the case studies that were conducted to evaluate the framework in practice. In Chapter 7, the design and creation process of the framework is reflected upon, as well as listing the scientific contributions of this work, threats to validity, limitations, and opportunities and future work. Finally, in Chapter 8 the concluding reflections on this study's research questions are summarized.

# Chapter 2

# Research Approach

This chapter describes the research approach which is to be adhered to over the course of this thesis. First, the reasoning for choosing to follow the Design Science Research (DSR) methodology is elaborated upon. This is followed by a description of the research design that guides this thesis. Next, the three main research methods this thesis consists of are described: (1) *a literature study* to identify, collect and compile all relevant knowledge in relation to the object of study, as well as populating the first version of the API management assessment tool or framework; (2) *expert interviews* to evaluate the first preliminary version of the API management assessment tool or framework; and (3) *case studies* to validate and test the final version of the tool or framework in practice. A visualization of the research approach, including the research methods it comprises, may be seen in the form of a Process Deliverable Diagram (PDD). This PDD can be found in Appendix A, and consists of two integrated diagrams. The first of which, located on the left side of the PDD, depicts the research process as based on an UML activity diagram. The PDD's right side shows the deliverables that were produced as a result of executing the various processes this research consists of, and are based on the UML class diagram (van de Weerd & Brinkkemper, 2009).

## 2.1 Design Science Research

In their work, Hevner, March, Park, and Ram (2004) present the two main paradigms that characterize much of the research performed in the Information Systems discipline:

> *Behavioural science:* Seeks to develop and verify theories that explain or predict human or organizational behavior.
>
> *Design-science:* Seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts.

Considering that the goal of this thesis is to create a new artifact that enables organizations to assess, evaluate and improve upon their API management related business processes, this work is categorized under the design-science paradigm. In order to create the aforementioned artefact, the Design Science Research methodology (DSR), as introduced by Hevner et al. (2004), has been adopted. This methodology involves the creation and evaluation of IT artifacts intended to solve identified organizational problems, which aligns with the intended goal of this thesis. In their work, Hevner et al. (2004) introduce the Information System research framework as part of the DSR methodology. Figure 2.1 depicts the application of this framework to

the context and scope of this thesis. This framework is composed of three elements, resulting in a conceptual representation of the research at hand. These elements comprise the environment, knowledge base and IS research. The environment defines the problem space in which relevant phenomena with regards to the research are located, and is composed of people, organizations and technology. Together, these define the business need or problem, as perceived by the researcher (Hevner et al., 2004). By framing research activities to adequately address business needs, research relevance is ensured. The knowledge base provides the researcher with the raw materials, such as literature, from and through which IS research is accomplished. This knowledge base is composed of foundations and methodologies, resulting in applicable knowledge. Using business needs and applicable knowledge as input, IS research is then able to be carried out. When adhering to the design science methodology, conducting IS research consists of building and evaluating artifacts that are designed to meet the identified business needs.

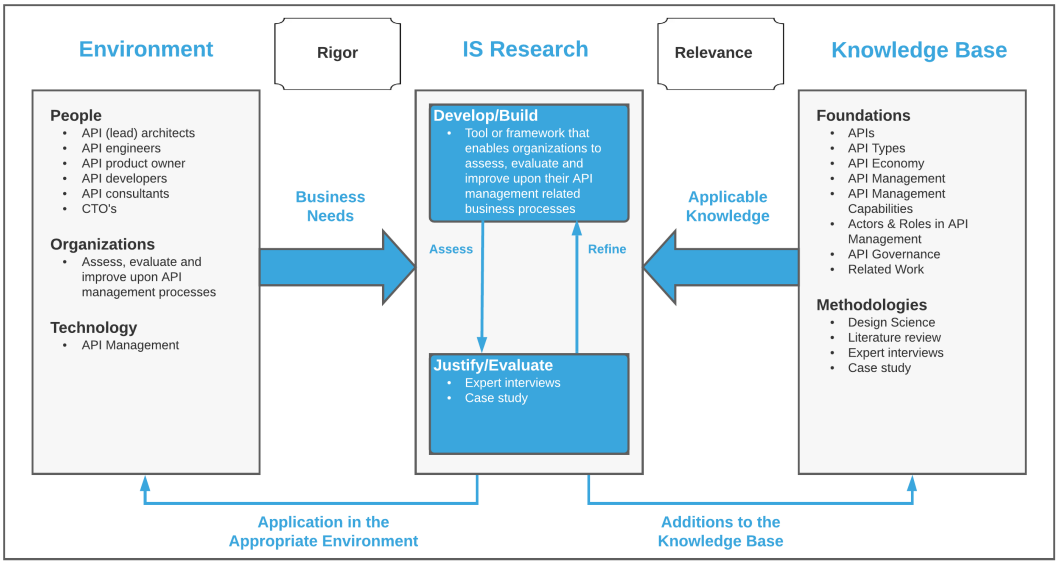



FIGURE 2.1: Information Systems research framework (Hevner et al., 2004), as applied to this research.

### 2.1.1 Research Methods

As part of the process of designing, evaluating and validating the artefact this study seeks to construct, answers to the research questions posed in Section 1.4 are formulated. The research methods used to do so are depicted in Table 2.1 below. First, a literature review is conducted to determine which type of framework or tool is best suited towards fulfilling this study's goal. Next, the results of the previously conducted Systematic Literature Review (SLR), which was summarized in Section 1.3, are used as input to initially populate the contents of the first version of the framework. Then, expert interviews are conducted in order to evaluate its *completeness, operational feasibility, ease of use, usefulness* and *effectiveness*. Lastly, the version of the framework which was produced as a result of the previous methods, is validated in a practical setting through an embedded, single-case study and multiple case studies. As a result, the framework or tool's *operational feasibility, ease of use, usefulness*

and *effectiveness* will be evaluated in practice. These criteria are described in further detail in Table 2.3.

TABLE 2.1: Research methods used to answer the SQs.

| Method | SQ1 | SQ2 | SQ3 | SQ4 |
|---|---|---|---|---|
| Literature Review | ✓ | | ✓ | |
| Systematic Literature Review | | ✓ | | |
| Expert Interviews | | ✓ | ✓ | |
| Case Studies | | | | ✓ |

## 2.2 Research Design

In order to construct the research design that guides this thesis, the Information Systems research framework by Hevner et al. (2004) has been combined with Polya (2004)'s problem-solving cycle. The resulting research cycle is depicted in Figure 2.2 below, and comprises the following key phases:

*Problem statement:* This first phase of the research cycle aims to adequately describe, identify, scope and frame the problem at hand. As a result of doing so, a problem statement is produced.

*Knowledge analysis:* The key objective of this phase is to identify, collect and compile all relevant knowledge in relation to the object of study. These activities yield an overview comprising all relevant subjects.

*Solution design:* This phase is targeted towards designing one or more candidate solutions, using knowledge extracted from the knowledge base as input. Next, the most appropriate and suitable candidate solution is selected with regards to the identified problem.

*Case study implementation:* In this phase, the previously selected candidate solution is applied in practice, as part of case studies.

*Solution Evaluation:* As part of this final phase, the implemented solution is evaluated by verifying whether the problem has been solved as a result of the implementation.

### 2.2.1 Problem Statement

This phase aims to adequately describe, identify, scope and frame the problem at hand. This is accomplished by performing an exploratory literature research, using the body of literature that was composed as a result of the Systematic Literature Review (Mathijssen et al., 2020a) as a starting point. This initial exploratory literature study ensured clear scoping of the domain. Furthermore, blog posts, consultancy reports and surveys were identified to highlight the identified problem. Additionally, initial potential candidate solutions were identified. Combined, these activities have resulted in the composition of the Problem Statement (Section 1.1) of this work.

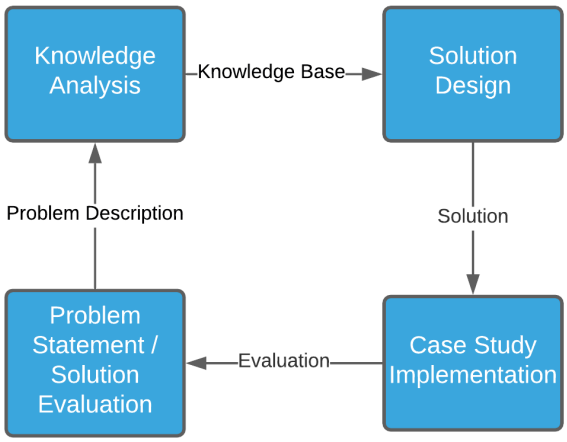



FIGURE 2.2: The research cycle, consisting of four phases, as combined from Hevner et al. (2004) and Polya (2004).

### 2.2.2 Knowledge Analysis

The key objective of this phase is to identify, collect and compile all relevant knowledge in relation to the object of study. In order to do so, the body of literature produced as a result of the Systematic Literature Review (as summarized in Section 1.3) was used as the starting set, on which the snowballing method was subsequently applied. Wohlin (2014) defines snowballing as follows:

> *Snowballing:* Refers to using the reference list of a paper or the citations to the paper to identify additional papers.

Wohlin (2014) notes that snowballing is particularly useful for extending a systematic literature study, considering that new studies almost certainly must cite at least one paper among the previously relevant studies or the systematic study already conducted in the domain. By iterative repetition of the snowballing method on the body of literature, all relevant concepts in the scope of this research were identified and composed. The main areas of interest in the scope of this research are as follows:

- *Application Programming Interfaces (APIs)*
- *API Types*
- *API Economy*
- *API Management*
- *API Management Capabilities*
- *Actors & Roles in API Management*
- *API Governance*

Additionally, all of the existing frameworks, tools, models, reports and case studies that that are related to the scope and goal of this study are summarized and evaluated.

### 2.2.3 Solution Design

As part of this phase, the design of candidate solutions based on the information gathered during the previous knowledge analysis phase is described. First, related work and literature on API management assessment tools and frameworks, which are described in Chapter 3, is used as input to inspire the candidate solution. By comparing and analyzing several of these related tools and frameworks, as well as suitable design science artefacts, it is decided upon which type of framework or tool forms the most suitable basis for solving the problem outlined in the problem statement. This process is described in Chapter 4. In doing so, SQ1 will be answered. After having done so, the tool or framework is constructed by using results of the previously conducted SLR (as described in Section 1.3) as input to initially populate it. This process forms a basis towards partially answering SQ2, and will be described in Chapter 4.

### 2.2.4 Solution Evaluation

The goal of this phase is to evaluate and validate the first preliminary version of the API management assessment tool or framework which was designed as a result of the previous phase. In order to successfully communicate the results of this evaluation process, Shrestha, Cater-Steel, and Toleman (2014)'s linear logic model will be used. This model provides a consistent reporting structure to communicate Design Science Research evaluation by presenting a unified view of (1) inputs in terms of the artefact to evaluate; (2) discussion of participation and activities to clearly explain the evaluation process; and (3) outcomes from the evaluation in terms of immediate findings, their discussion and long-term impacts. In applying this reporting structure to this research, the model has been modified as depicted in Figure 2.3.

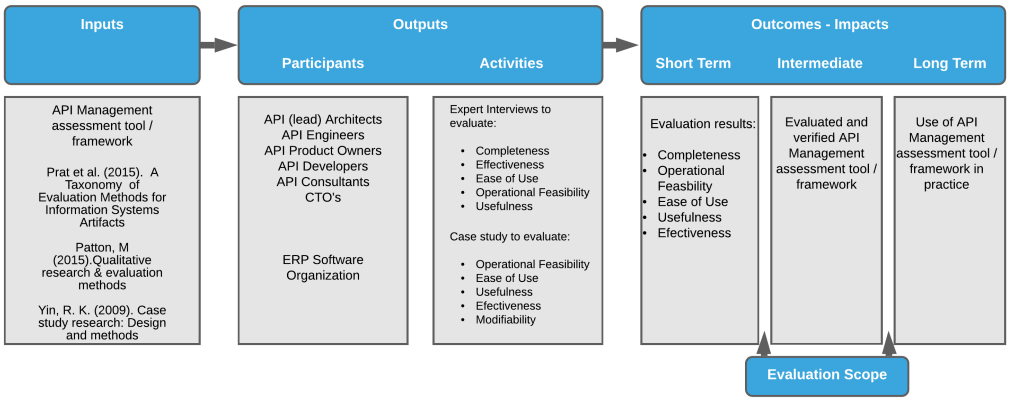



FIGURE 2.3: Shrestha et al. (2014)'s evaluation reporting model, as applied to this research.

In order to evaluate and validate the first version of the API management assessment tool or framework, two main methods will be utilized; expert interviews and a case study. These methods were selected due to the fact that the main objective of this research is to develop a tool or framework that is grounded and verified in both literature as well as the software industry. Using insights and feedback gained from these approaches, the first version of the tool or framework will be iteratively

improved upon by making incremental adjustments to it when needed, producing new versions throughout the process (Hevner & Chatterjee, 2010). Changes to the current version will be applied in the event where there is deemed to be a sufficient amount of evidence to warrant completing a new increment.

### Evaluation Strategy

Utilizing expert interviews and a case study as methods for evaluation were selected due to the fact that the main objective of this research is to develop a tool or framework that is grounded and verified in both literature as well as the software industry. In order to verify the appropriateness of these approaches in relation to the scope and goal of this research, the DSR evaluation strategy selection framework and DSR evaluation method selection framework by Venable, Pries-Heje, and Baskerville (2012) were investigated. Next, the strategic evaluation framework by Venable et al. (2012) was applied to this research, as can be seen in Table 2.2.

TABLE 2.2: Strategic evaluation framework (Venable et al., 2012), as applied to this research.

| Research Phase | Evaluation Type | Evaluation Method | Evaluation Criteria | Evaluation Methodologies |
|---|---|---|---|---|
| Solution design | Ex-post, artificial | Alignment with DSR and SLR guidelines | Completeness | DSR methodology (Hevner et al., 2004) and guidelines for conducting SLR's (Kitchenham & Charters, 2007; Okoli, 2015) |
| Solution evaluation | Ex-post, artificial | Expert interviews | Completeness, operational feasibility, ease of use, usefulness, effectiveness | Qualitative research & evaluation methods (Patton, 2014) and Taxonomy of Evaluation Methods (Prat, Comyn-Wattiau, & Akoka, 2015) |
| Solution evaluation | Ex-post, artificial | Case studies | Operational feasibility, ease of use, usefulness, effectiveness | Case study research: Design and methods (Yin et al., 2003) and Taxonomy of Evaluation Methods (Prat et al., 2015) |

In order to be able to interpret the results from the aforementioned evaluation methods, a set of evaluation criteria is utilized. This set consists of a collection of criteria which are presented by Prat et al. (2015) and were subsequently modified to match the scope and goal of this research, as can be seen in Table 2.3. By using these criteria, the artefact is able to be evaluated in terms of its completeness, ease of use, effectiveness, operational feasibility, and usefulness.

### Expert Interviews

As a first step towards evaluating and improving upon the first version of the API management assessment tool or framework, expert interviews will be conducted. Interviews provide researchers with rich and detailed qualitative data (Rubin & Rubin, 2011) and are an effective way to evaluate artifact designs at an early stage (Wieringa, 2014). In the context of this research, expert interviews will be utilized as an instrument to answer research questions SQ2 and SQ3, by gather feedback and satisfying the completeness criterion as described in Table 2.3. In doing so, it will be ensured that the API management assessment tool or framework comprises all relevant (best) practices, capabilities and features the topic consists of. This will be accomplished by presenting experts with the first version of the tool or framework,

TABLE 2.3: Evaluation criteria and their corresponding definitions, as well as the evaluation tools used.

| **Criteria** | **Definition as presented by Prat et al. (2015)** | **Modified definition in scope of this research** | **Evaluation Tool** |
|---|---|---|---|
| Completeness | The degree to which the structure of the artifact contains all necessary elements and relationships between elements. | To what degree does the API management assessment framework or tool contain all the (best) practices, capabilities and features API Management is comprised of? | Open-ended questions |
| Ease of Use | The degree to which the use of the artefact by individuals is free of effort. | To what degree is it free of effort for an organization to self-assess and evaluate their degree of maturity with regards to API management, using the API management assessment tool or framework? | 5-point Likert scale |
| Effectiveness | The degree to which the artifact achieves its goal in a real situation. | To what degree does the API management assessment tool or framework succeed in aiding organizations to improve upon their API management-related business processes? | 5-point Likert scale |
| Operational Feasibility | Evaluates the degree to which management, employees, and other stakeholders, will support the proposed artifact, operate it, and integrate it into their daily practice. | To what degree are experts expecting organizations to utilize the API management assessment tool or framework in practice? | 5-point Likert scale |
| Usefulness | The degree to which the artifact positively impacts the task performance of individuals. | To what degree does the API management assessment tool or framework provide organizations with valuable and useful insights regarding their API management-related business processes? | 5-point Likert scale |

which was initially populated with results of the previously conducted SLR. By using open-ended and 5-point Likert scale questions, the structure and contents of this first version of the API management assessment tool or framework will then be evaluated.

In order to ensure the reliability and effectiveness of the interviews, the interview protocol refinement (IPR) framework by Castillo-Montoya (2016) is utilized. This framework comprises a four-phase process to develop and fine-tune interview protocols. IPR's four phases include ensuring interview questions align with this study's research questions, composing an interview protocol in order to create an inquiry-based conversation, having the protocol reviewed by others, and piloting it. By doing so, the reliability of the interview protocol is enhanced, increasing the quality of the obtained data as a result, and increased congruency with the aims of this study (Jones, Torres, & Arminio, 2014).

**Phase 1** focuses on the alignment between interview questions and research questions. This alignment will increase the utility of interview questions in the research process, while ensuring their necessity for the study. By mapping interview questions onto research questions, any existing gaps regarding missing questions are identified as well as unnecessary questions being removed. This mapping process is accomplished through the usage of a matrix, which can reviewed in Appendix B.

**Phase 2** entails seeking to strike a balance between inquiry and conversation, resulting in an inquiry-based conversation by asking questions for specific information related to the aims of this study (Castillo-Montoya, 2016; Patton, 2014). This is accomplished by constructing the interview protocol so that: (1) interviews questions

are formulated in a clear and easily understandable manner when compared to the research questions; (2) the conversation is structured in a natural way, by including introductory and transition questions as well as logically ordering questions; (3) a variety of questions that cover all relevant topics and aspects is included; (4) a script is adhered to, which included possible follow-up and prompt questions.

**Phase 3** is concerned with receiving feedback on the developed interview protocol. The purpose of obtaining feedback on the interview protocol is to enhance its reliability and trustworthiness as a research instrument (Castillo-Montoya, 2016). Through this feedback, information is obtained regarding the degree to which participants understand the interview questions and whether their understanding is close to what the researcher intends or expects (Patton, 2014). For this research, feedback will be obtained through having research team members examining the first version of the interview protocol for structure, length, writing style, and comprehension. In order to aid with this, research team members are provided with an activity checklist for close reading of an interview protocol, as developed by Castillo-Montoya (2016). This checklist may be reviewed in Appendix B.

**Phase 4** is aimed towards finalizing the interview protocol, by piloting the refined interview protocol with people who mirror the characteristics of the sample to be interviewed for the actual study (Maxwell, 2012). However, as Castillo-Montoya (2016) remarks, researchers may not have the time to engage in a piloting phase, making phase 3 become even more crucial towards refining the interview protocol. Considering that this is the case for this research, an iterative refinement method is opted for instead, in addition to utilizing the refinement checklist as part of the previous phase. After each interview, lessons learnt will be incorporated and used as input to further improve upon and refine the interview protocol. The current interview protocol may be reviewed in Appendix B.

**Expert Selection Criteria**

As mentioned earlier in this work and as was made evident by the results of the previously conducted SLR, available literature on the topic of API management is relatively scarce. Because of this, experts on this topic need to be consulted in order to verify that the contents of the API management assessment framework or tool are complete and correct. In order to do so, it has to be ensured that the selected experts are experienced and knowledgeable regarding the subject. This is achieved through the usage of the purposive sampling technique, which refers to the deliberate choice of a participant due to the qualities the participant possesses (Etikan, Musa, & Alkassim, 2016). For this research, a sub-method of purposive sampling is used; expert sampling. Expert sampling involves the calling for experts in a particular field to be the subjects of the purposive sampling (Etikan et al., 2016). This sampling method is useful when the research is suffering from a lack of observational evidence and is effective when investigating new areas of research (Etikan et al., 2016).

In order for the expert interviews to produce useful results, participants should be experienced and knowledgeable with regards to the topic of API management. As such, participants should adhere to the following requirements: (1) potential participants must indicate to be knowledgeable on a minimum of 75% of the first version of the tool or framework's topics; (2) potential participants must have a minimum of 3 years of experience with either consuming, developing, integrating, providing,

versioning, monitoring or managing APIs; (3) potential participants must be working at an organization as an architect, developer, engineer or product owner as part of a team working with APIs, or as a CTO, IT consultant or any comparable role. In order to establish whether the potential participant satisfies the first requirement, a short preliminary survey will be sent out, requesting the potential participant to indicate the degree of knowledge they possess with regards to the topic. This survey may be reviewed in Appendix B. This will be accomplished by providing them with a short list of all the main elements that are contained in the first version of the API management assessment tool or framework, accompanied by a description. After having read these descriptions, potential participants will then be asked to denote their knowledge on the individual subjects through the use of 5-point Likert scale questions. In order to verify the second and third requirement, potential participants will be asked to fill out their experience and current role within their organization. A large portion of potential participants was identified and contacted through the usage of the case company's, which will be described in detail as part of the next subsection of this chapter, partner network. Additional potential participants were identified and contacted through the usage of the first project supervisor's *LinkedIn* profile network. Furthermore, potential participants on both the API provider and consumer side will be contacted. By doing so, it is ensured that knowledge stemming from both perspectives of API management is incorporated in the API management assessment framework.

On initial contact through e-mail with the purpose of gauging interest, potential participants were provided with detailed information describing the first version of the API management assessment framework (Mathijssen, Overeem, & Jansen, 2020b). In case the potential participant showed interest in participating, he or she was asked to fill out the aforementioned short survey. Then, if the potential participant passed all imposed requirements for participation, participants were asked to read an information sheet and sign an informed consent form. These forms can be found in Appendix B and were created using a template by TU Delft (2018) and guidelines by Utrecht University (2020).

#### Case Study Implementation

As mentioned earlier, a case study will be conducted in order to evaluate and validate the *operational feasibility, ease of use, usefulness* and *effectiveness* of the API management assessment framework or tool, answering SQ4 in a result. Venable et al. (2012) state that the utilization of case studies as part of the DSR methodology supports the goal of evaluating the effectiveness of an artefact's design. Furthermore, Runeson and Höst (2009) argue that the case study approach is a suitable research methodology for conducting software engineering research, considering that it studies contemporary phenomena in a practical context. The structure of this case study is supported by guidelines for conducting case studies by Runeson and Höst (2009) and is based on five key phases, as formulated by Yin et al. (2003).

**Phase 1** concerns the design of the case study. According to Yin et al. (2003), the case studies to be conducted as part of this research is classified as an *embedded, single-case study*, as well as *multiple case studies*. This classification is based on the fact that while this case study will take place at a single case company, multiple units of analysis will be involved. Furthermore, the framework or tool is evaluated with

multiple other case companies. The selected case companies are described in Chapter 6.1.1 and Chapter 6.1.2.

**Phase 2** addresses the preparation for data collection. This involves the creation of a case study protocol. This protocol, which may be found in Chapter 6.1.3, serves as a guide when conducting the data collection, and prevents failing to collect data that was planned to be collected (Runeson & Höst, 2009). Furthermore, it improves the reliability of the case study research, and ensures the researcher remains focused on the subject of the case study (Yin et al., 2003). The case study protocol will be reviewed and validated by relevant stakeholders within the case company, which among other things, will ensure that the risk of missing relevant data sources is lowered (Runeson & Höst, 2009).

**Phase 3** involves the actual data collection process. During this phase of the case study, data is collected and combined from different sources. Doing so is in compliance with guidelines formulated by Runeson and Höst (2009), who states that it is paramount to use several data sources in a case study in order to limit the effects of one interpretation of a single data source. As part of his work, Yin et al. (2003) discerns several data source types. For this case study, *interviews* with relevant stakeholders will be conducted. This instrument is used in order to gain insight into the manner in which API management activities and processes are implemented in the case company. Additionally, stakeholders will be asked to evaluate the framework or tool with regards to the evaluation criteria listed in Table 2.2.

**Phase 4** concerns the analysis of the data collected as part of the previous phase. In order to do so, data originating from different sources will be combined and analyzed. Based of the results of this analysis, the framework or tool will be adapted and improved upon.

**Phase 5** involves reporting on the findings from the case study. This will be done by distilling findings from the previously performed analysis into an advisory case study report. Furthermore, all findings are reported on in Chapter 6.2.

# Chapter 3

# Literature Review

In this chapter, all related and relevant literature in the scope of this research is described. First, the core topic of this work, APIs, is described, followed by the various types that may be discerned among APIs. Next, the API economy is elaborated upon, followed by a general overview of API management. Then, the topic of API management is dissected by discussing the various capabilities the discipline comprises. Next, the various actors, roles and positions that exist in the domain of API management are presented, after which the API governance approach is presented. Lastly, existing frameworks, tools, models, reports and case studies that are related to the scope and goal of this study are summarized and evaluated.

## 3.1 Application Programming Interfaces (APIs)

An Application Programming Interfaces (API) is a software-to-software interface that defines a contract for applications to communicate with one another over a network, without the need for any user interaction (De, 2017). By adhering to this contract, APIs expose services or data which are provided by a software application in the form of a set of predefined resources. These resources may consist of methods, cleansed data, images, videos, documents, or functionality that performs complex calculations or data processing based on inputs (Stylos (2009); L. Weir (2019). By using these resources, other applications are able to access the data or services without having to implement the underlying objects and procedures. APIs are a central element in many modern software architectures, considering the fact that they provide high-level abstractions that facilitate programming tasks, support the design of distributed and modular software applications and the re-usage of code (Robillard, 2009). As Myers and Stylos (2016) point out, all modern software makes heavy use of APIs. Nowadays, instead of programming desired functionalities from scratch, the fundamental task of software developers often is to stitch together functionality that existing APIs provide (Stylos, 2009).

## 3.2 API Types

Generally speaking, APIs may be classified into two categories. The first of these main categories encompasses the protocol that is used in enabling client-server communication. Two of such protocols exist, both of which serve different purposes and offer varying benefits. These protocols, called REST and SOAP, are elaborated upon in further detail below. The second main category APIs may be categorized into, concerns the intended users and those that have ownership of the API. These ownership types are subdivided into public, internal, and partner APIs.

### 3.2.1 Web (Service) APIs & Protocols

Web services aim to simplify processes by defining a standardized mechanism to describe, locate, and communicate with online applications (Curbera et al., 2002). One type of APIs, called web service APIs, utilize these web services to offer a systematic and extensible approach towards integrating services into (existing) applications (Espinha, Zaidman, & Gross, 2014). In relation to web services, web APIs may best be regarded to be the face of a web service, directly listening and responding to client requests (Masse, 2011). In the past, web service APIs had become synonymous to SOAP protocol-based web service APIs. In recent years however, the emphasis has been moving towards usage of REST protocol-based communication (De, 2017). With the advent of REST and due to its popularity, RESTful web services and web APIs are nowadays usually commonly referred to simply as APIs (De, 2017). However, SOAP and REST are protocols that both have different appliances and advantages in terms of performance. In order to fully illustrate this, the REST and SOAP protocols are explained in further detail below.

#### Representational State Transfer (REST)

REST is a very popular web API protocol which defines a set of constraints aimed towards improving the performance, availability and scalability of web-based distributed systems (Fielding, 2000). REST is based on the traditional client-server paradigm, which holds that requests issued by clients are executed by a server (Salvadori & Siqueira, 2015). REST defines a uniform interface for system components based on a set of four constraints: resource identification, handling resources through representations, self-described messages and Hypermedia as the Engine of Application State (HATEOAS).
The resources the first two constraints are concerned with are abstractions of information, which are handled by applications adhering to the REST architecture, and must be identifiable as well as accessible through a generic interface. However, resources are not directly accessed. Alternatively, they are viewed through a representation, which is a snapshot of the state of a resource at a given point in time (Wilde & Pautasso, 2011).
In order for services to adhere to the third constraint, behavior must be stateless. This entails that the server does not log information regarding issues requests by clients, which is key in improving the scalability of a system. In order to achieve stateless behavior, self-described messages must be utilized. These messages are requests that contain all the information required for being processed by the server. The body of these messages can be structured in the XML or JSON format. The latter is often preferred for mobile and web apps using RESTful services, considering that it is a lightweight format, increasing performance as a result (De, 2017). The last constraint, called HATEOAS constraint, holds that client interaction must be guided through the usage of hypermedia control. These are controls which provide mechanisms for querying, storing and handling information on resources (Salvadori & Siqueira, 2015).

#### Simple Object Access Protocol (SOAP)

SOAP (Simple Object Access Protocol) is an established web API protocol that is often preferred for service interactions within an organizations. Other systems interact with the web service in a manner prescribed by its description using SOAP messages (De, 2017). These messages are transported using protocols such as HTTP,

SMTP or FTP. At its core, the SOAP message structure consists of an SOAP envelope, which contains the SOAP headers and the SOAP body, which in turn contains the actual information to be sent (De, 2017). SOAP is based on the standard XML format, which is designed specifically to transport and store structured data. It incorporates security and authorization into its protocol and can be used to enforce a formal contract between the client and server (Patni, 2017). However, SOAP-based web services are considered to be relatively heavyweight, because additional processing of extra SOAP elements in the payload is required (De, 2017). Because of this, it should not be utilized when bandwidth capacity is limited (Patni, 2017).

### 3.2.2 API Ownership Types

Generally speaking, APIs may be classified into three categories when examining their nature in terms of accessibility and ownership. These categories are visualised in Figure 3.1 below, and described in further detail thereafter.

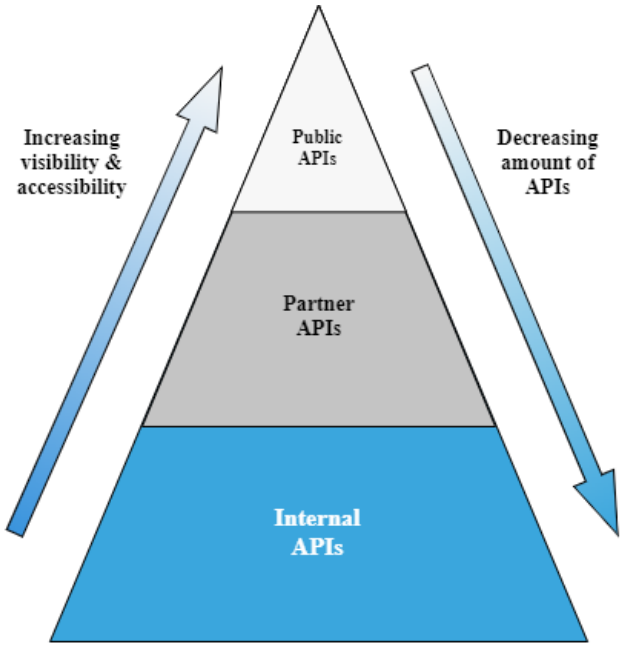



FIGURE 3.1: API ownership types, as related to their degree of accessibility, visibility and amount.

- *Public APIs*: The interface of a public API is designed with the purpose of being accessible by a wide developer community (De, 2017). Public APIs can be accessed by internal app developers within an organization and other users with minimal restrictions, which may require registration, or may be completely open (Castellani and Dorairajan (2020); Jacobson, Woods, and Brail (2011). By being open to a wider audience of app developers, public APIs are able to assist an organization in adding value to its core business through innovation. Additionally, public APIs also lead to an increase in the usage of company assets, and add business value without direct investment in app development (De, 2017). When compared to the two subsequent API ownership types below, the offering of public APIs is considered to be the smallest.

- *Internal APIs*: In contrast to open APIs, internal APIs are designed to be hidden from external users and are intended for internal app integration or for mobile and web app development with the goal of internal consumption (De, 2017). The use and reuse of internal APIs can enhance an organization's productivity, efficiency, and agility (Mulesoft, 2020). Additionally, the internal use of a company's private APIs for business transformation can often derive more business benefits when compared to public APIs (De, 2017). Furthermore, internal APIs are considered to form the 'large underwater mass of the iceberg', due to their abundance and popularity in terms of their usage when compared to visibly exposed public APIs.
- *Partner APIs*: Partner APIs are mostly intended for business-to-business integration with partners of the organization and are usually consumed by partners while adhering to contractual agreements (De, 2017). Similarly to internal APIs, they are intended for private use, and are not exposed to the general public. Furthermore, partner APIs share open API's purpose of enabling communication beyond the boundaries of an organization.

In addition to mapping these ownership types to their degree of visibility and accessibility, observations may be made regarding their popularity across the software industry. To this end, a survey conducted by Postman has found that 52.8% of respondents state that the majority of APIs they work with are internal (Postman, 2019). Furthermore, 28.4% of respondents utilize APIs that were exclusively shared among integration partners, and only 18.8% of respondents work with public APIs.

## 3.3 API Economy

In recognizing that APIs that are part of an ecosystem are more valuable than when they exist in isolation, organizations have started to share their internal ICT business assets in the form of APIs, both across internal organizational units as well as to external third parties (Bonardi, Brioschi, Fuggetta, Verga, & Zuccalà, 2016; Vukovic et al., 2016). Doing so results in unlocking additional business value through the creation of new asset classes, in addition to being able to react and restructure quicker compared to when APIs are exclusively used internally, as well as being able to outsource capabilities via APIs (Bonardi et al. (2016); Medjaoui et al. (2018). Nowadays, the trend described above is widely referred to as the *API Economy*. The API Economy may best be described as the commercial exchange of business functions, capabilities, or competencies as services using web APIs (Holley et al., 2014). In the API economy, organizations are offering access and the ability to recombine their digital services and products for novel value creation.

Over the past decade, the API economy has grown exponentially, with most leading organizations offering some form of APIs for accessing their services and products (Basole, 2019). For organizations to succeed in the API economy, a shift away from traditional approaches to delivering services needs to be made, towards to a product-centric approach whereby the main goal is to make an API product successful and profitable (L. A. Weir, Bell, Carrasco, & Viveros, 2015). According to Holley et al. (2014), participating in the API economy yields the following benefits for organizations:

- Growing the customer base by attracting customers to products and services through API ecosystems.

- Driving innovation by capitalizing on the composition of different APIs, including internal, partner or public APIs.
- Improving the time-to-value and time-to-market for new products.
- Improving integration with web APIs.

### 3.3.1 API Value Chain

As mentioned earlier, in the API economy, organizations are offering access and the ability to recombine their digital services and products for novel value creation. Value is created as part of the *API Value Chain*, which is depicted in Figure 3.2 below. The actors involved in the API value chain are summarized in Table 3.1 below.

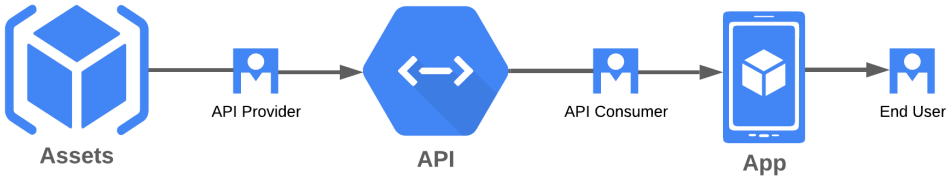



FIGURE 3.2: The API Value Chain, consisting of 3 actors and 3 assets.

First, the API provider identifies business assets, including their corresponding value, and decides to make it available for others to use (De, 2017). These business assets can take the form of any data or business functionality and can range from product catalogs, to customer information, to payment services or Twitter posts (De, 2017; Jacobson et al., 2011). Ultimately, the value and usefulness of these business assets determine the success of the API. After having identified assets, an API is created so that they may be exposed. The API provider is responsible for designing the API so that the intended audience may easily use it. Oftentimes, the owner of the exposed business assets is the API provider, in which case all benefits are reaped by the asset owner (Jacobson et al., 2011). After the API has been designed and exposed, the API consumer may then assess and incorporate the API's functionalities into an application. These applications may take the shape of mobile apps or web apps, and should be distributed and made available to the end user in order to add value to the organization (De, 2017).

TABLE 3.1: Basic actors as part of the API value chain.

| Actor | Description |
|---|---|
| API Provider | The API provider is the entity that supplies business assets and services through an API, by designing and exposing it for API consumers to use (De, 2017; Jacobson et al., 2011); Farrell (2009). |
| API Consumer | Often referred to as the API developer, an API consumer is an entity that calls the API to use the business assets and services it provides (Farrell, 2009). Typically, the API consumer utilizes the API to develop mobile apps, software applications, websites or web apps by integrating functionalities that are offered by the API into them (Crowe, 2018). |
| End User | End users are the users of the final product, which take the shape of an mobile app, software application, website or web app. End users may use this app on their mobile devices, smartphones, tablets, desktops and so forth (De, 2017). |

## 3.4 API Management

In order to fully profit from the API economy, after an API has been designed, created and exposed, organizations need to properly manage their API program so that consumers may easily integrate it into their applications. In its essence, this is accomplished by performing activities such as providing helpful documentation, controlling and monitoring access to the API, as well as monitoring and analysing its usage. However, API management is a broad and diverse topic that is interpreted and defined in varying ways among the research community, as was found as part of the previously conducted SLR (Mathijssen et al., 2020a). A basic definition is given by Coste and Miclea (2019), defining the topic as follows: *"API management is the practice an organization implements to manage the APIs they expose. This is done internally or externally so it makes sure that the APIs are consumable, secure, and available to consumers in conditions agreed upon in the terms of use of APIs"*. This definition highlights the core activities of API management, which is for an API provider to expose APIs for internal and external parties to consume, in a secure and agreed upon manner.

Another basic definition of API Management is given by Vijayakumar (2018), stating that: *"API Management is a solution encompassing the collections of tools used to design and manage APIs, referring to both the standards and the tools used to implement API architecture"*. One of the main tools that are mentioned as part of this definition are API management platforms. De (2017) defines API management platforms as follows: *"An API management platform helps an organization publish APIs to internal, partner, and external developers to unlock the unique potential of their assets. It provides the core capabilities to ensure a successful API program through developer engagement, business insights, analytics, security, and protection"*. Among the 24 unique definitions found of API management, which were identified as part of the previously conducted SLR (Mathijssen et al., 2020a), management platforms are mentioned 6 times. Combined with the observation that the above definition by De (2017) is adopted by three other authors, it appears that many authors of literature related to API management consider API management platforms to be an integral component of the topic. As such, this component was incorporated into the definition of API management that was formulated as a result of the aforementioned SLR. This definition is as follows: *"API Management is an activity that enables organizations to design, publish and deploy their APIs for (external) developers to consume. API Management encompasses capabilities such as controlling API lifecycles, access and authentication to APIs, monitoring, throttling and analyzing API usage, as well as providing security and documentation*. As was mentioned in Section 1.3, this definition was composed as a result of a key-term frequency analysis using all definitions that were encountered across literature on the topic as input.

There are several commercial API management platforms available on the market, which are considered to be highly regarded enterprise tools (L. Weir, 2019). Organizations may choose from a plethora of commercially available API management platforms, the most prominent of which, according to Forrester Research (2020)'s latest evaluation of API management solutions, include Software AG, IBM API Connect, Google's Apigee, Oracle's WSO2 and Axway's AMPLIFY API Management. By utilizing such an API management platform, organizations are able to increase business outreach across digital channels, drive partner adoption, monetize digital assets, and provide analytics to optimize investments in digital transformation (De,

2017). Typically, most API management platforms offer services of three major types, as is depicted in Figure 3.3 below. These service types are elaborated upon in further detail in the following subsections.

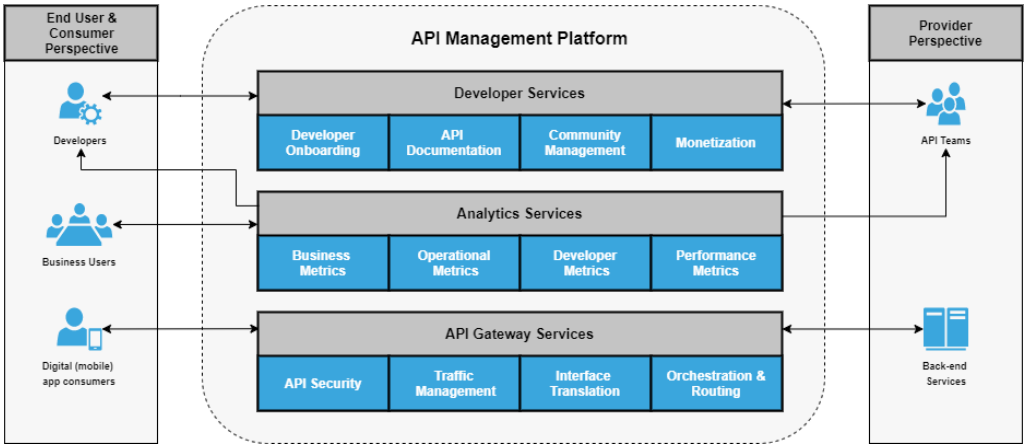



FIGURE 3.3: Common API management platform service types, as adapted from De (2017).

### 3.4.1 Developer Services

The main component that enables developer services as part of an API management platform is the *Developer Portal*. A developer portal is a customized web site that allows an API provider to provide services to the developer community, acting as a doorway to the organization's API program (De, 2017). Essentially, the developer portal is a content-management system that hosts all the supplementary resources for an API, which among other things, typically includes documentation functionalities and features, interfaces, usage guides and terms of use (Jacobson et al., 2011). Additionally, potential API consumers may explore and discover any APIs that have been published by API providers (Indrasiri & Siriwardena, 2018). Other core features of the developer portal include enabling new consumers of an API to sign up and register their applications to use the API, as well as fostering community engagement. This is achieved by offering developers a platform through which they may both communicate with one another as well as with the API provider. Communication may take place statically through documentation, or dynamically in the form of forums and blogs. Using such a platform, developers may share their experiences with API usage, raise support tickets, advice and best practices. In doing so, a truly dynamic community may be developed in order to enhance the user experience of the provider's APIs.

### 3.4.2 Analytics Services

Analytics services are a critical factor from both the technical as well as business perspective. These services provide the API provider with important information and business insights so that future decisions may be made. Information collected by analytics services includes monitoring of traffic from individual apps, operational metrics, API and app performance, and developer engagement metrics (De, 2017). Additionally, these capabilities provide the ability to troubleshoot problems and identify

bottlenecks (Indrasiri & Siriwardena, 2018). Furthermore, some API management platforms are able to visualize reports and metrics through reporting dashboards (Vijayakumar, 2018).

### 3.4.3 API Gateway Services

The main component that forms the heart of any API management platform is called the *API Gateway*. An API gateway is a server-based tool that is designed to reduce the cost of deploying APIs within a network, and enables secure, flexible, and reliable communication between the back-end services and digital apps (De, 2017; Medjaoui et al., 2018). Gateways are typically designed to be highly scalable, reliable, and secure, considering that improving the security of an API implementation is often the primary motivation for introducing a gateway into a system architecture (Medjaoui et al., 2018). Its main functionality is to help expose, secure, and manage back-end data and services as APIs, as well as to provide a framework to create a facade in front of the back-end services by intercepting all the requests that API consumers send (De, 2017; Indrasiri & Siriwardena, 2018). This facade intercepts and handles incoming API requests to enforce security, policies, validate data, verify the identity of the consumer, limit the number of allowed calls based on rules, transform messages, throttle traffic, and finally route it to the back-end service (De, 2017; Vijayakumar, 2018; L. Weir, 2019). Optionally, gateway services may be extended in order to carry out tasks such as orchestrating requests between multiple back-end services or composing multiple business functionalities (De, 2017; Indrasiri & Siriwardena, 2018).

## 3.5 API Management Capabilities

The topic of API management is broad in its scope and is a discipline that, often supported by an API management platform, comprises a variety of features and capabilities. However, an analysis conducted by Gámez Díaz, Fernández Montes, and Ruiz Cortés (2015) shows that API management platforms often differ in terms of the features they provide. This may include differences in possible security features, traffic management or pricing plans. However, there does not seem to be a clear consensus among authors on API management as to what the most common features offered by API management platforms are. As part of this literature review, four main books that discuss the topic of API management and the activities, capabilities and features it comprises, were identified. The first of these, which was written by Jacobson et al. (2011), is aimed towards providing business executives with a roadmap for them to incorporate APIs into their business. In doing so, strategies, business models, and API design, community engagement and security considerations are discussed. Similarly, the work by Medjaoui et al. (2018) aims to support businesses that are starting off their API program. This is done by providing an API management framework that, among other things, describes ten 'pillars', each describing different aspects of the topic, as well as the various roles and responsibilities that are involved in guiding an API throughout its existence. A third book that focuses on providing businesses with advice for implementing an API strategy is that of L. Weir (2019). In this work, the community aspect of API management, ecosystems, organizational structures and architectural patterns are emphasized. Two other publications, those of Patni (2017) and Vijayakumar (2018),

approach the topic of API management from a more practical and technical perspective. This is accomplished by emphasizing and elaborating upon the actual design process of APIs, providing code samples, and describing the implementation and usage of API management platforms in practice. The last main book on API management that was identified, is that of De (2017). Compared to the work of authors mentioned earlier, this work is the broadest in scope and most complete in terms of capturing the topics API management consists of. Additionally, definitions of concepts related to API management that were formulated by this author are cited most often across all academic publications.

In addition to being the most cited, comprehensive and detailed work on the topic of API management, De (2017) presents a clear categorization of the capabilities API management consists of. As such, this categorization was used as a foundation to structure and subsequently describe the most common API management capabilities. When applicable, these were complemented and extended by using the work of the other books mentioned above as input. De (2017) proposes that the most common API management capabilities may be categorized into four groups, as shown in Figure 3.4 below. These capability categories are discussed in further detail in the subsequent subsections.

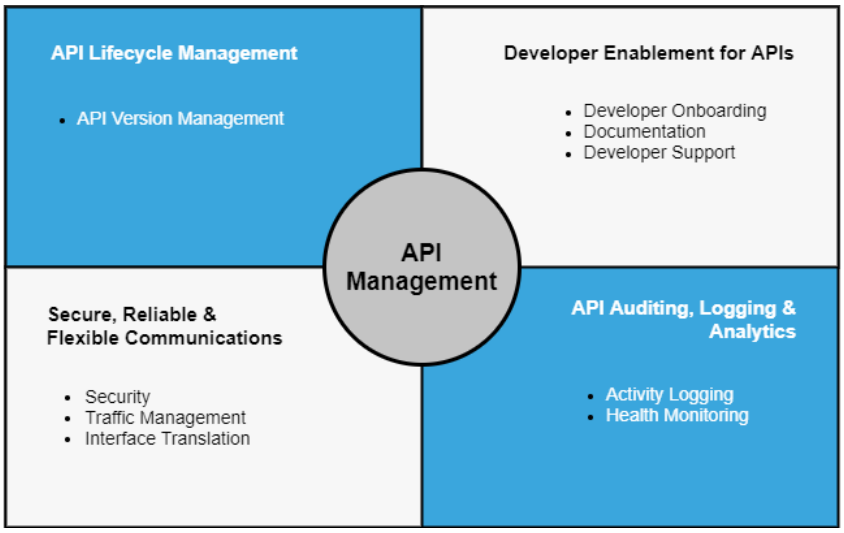



FIGURE 3.4: Four common API Management capabilities and the features they comprise, as adapted from De (2017).

### 3.5.1 API Lifecycle Management

Generally speaking, an API undergoes several stages over the course of its lifetime. In order to control and guide the API through these stages, API lifecycle management provides API providers with the capability to manage their API throughout its lifecycle and control how an API is developed and released to consumers (De, 2017). As mentioned earlier, the API lifecycle consists of several stages. Unfortunately, there does not seem to be a clear consensus among authors on work regarding API (lifecycle) management as to what stages the API lifecycle consists of. However, the core stages of the API lifecycle that are mentioned by most authors are visualized in Figure 3.5 below and involve the following:

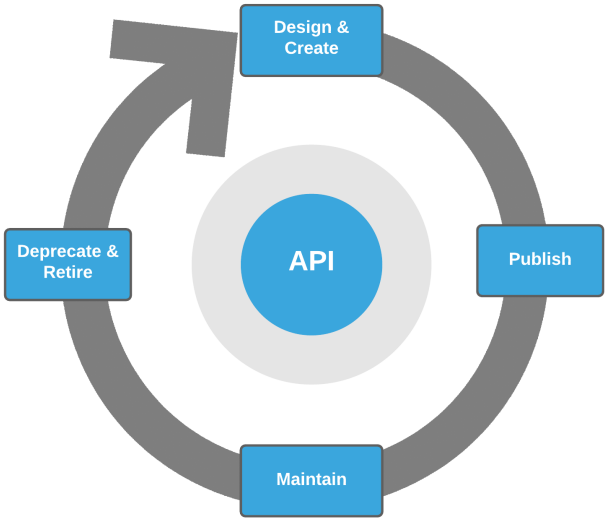



FIGURE 3.5: The core API Lifecycle, consisting of four stages.

- *Design & Create*: In this phase, the API team designs the interface for the API and possibly creates an API proxy to interact with the back-end services (De, 2017). Similarly to the API gateway, an API proxy acts as a facade to securely expose the back-end services to its consumers. Additionally, design decisions regarding aspects such as the authentication and authorization protocols that will be used should be addressed. After having done so, the actual creation and implementation of the API takes place. This involves the developers producing code and testing the API.

- *Publish*: Once an API has been designed and created, it must be published to an environment before it may be discovered and consumed by third parties (De, 2017). Ensuring APIs are discoverable is a fundamental aspect of API management. Additionally, relevant consumer-oriented documentation should be published alongside the API, ensuring that developers can reuse it, ultimately reducing development and operations costs (L. Weir, 2019). Typically, publication takes place within the developer portal so that consumers may easily access it.

- *Maintain*: After having been published, the API enters the maintenance stage, where it is likely to spend the largest portion of its lifecycle. Considering an API must continuously evolve by taking evolving consumer needs and expectations into account, it must undertake a series of iterations and changes (L. Weir, 2019). These may include actions such as bug fixes and modernization improvements, which constitute into an activity called version management (Medjaoui et al., 2018). Given its importance, API versioning are discussed in further detail in the following subsection.

- *Deprecate & Retire*: The lifecycle of an API ends with the retirement stage, which involves deprecating the API in a manner that aims to minimizes impact on existing consumers (L. Weir, 2019). There may be a plethora of reasons to retire or deprecate an API, which may include a lack of partner or third-party developer innovation, security concerns, decreasing demand or changes to operational

costs (Medjaoui et al., 2018; Patni, 2017). Retiring an API entails a different activity for each organization, as multiple strategies are possible. These may include removing all active API instances from production servers, or simply marking the API as deprecated and halting support as well as any future updates (Medjaoui et al., 2018).

As mentioned earlier, there does not seem to be a clear consensus among authors on work regarding API (lifecycle) management as to what stages the API lifecycle consists of. For example, in addition to the core stages described above, Medjaoui et al. (2018) suggests that an API enters the *realize* stage between the create and publish stages. They argue that during this stage, the API must meet its strategic objectives, which may include objectives such as reaching a certain amount of registered developers. L. Weir (2019) puts more emphasis on the design and create stage, by proposing and discerning three additional stages: *mock*, *try* and *configure*.

#### API Version Management

As mentioned earlier, an API continuously evolves during the maintenance stage. However, because API linkages are prone to breaking, problems generally occur when APIs start to evolve over time (Xavier, Brito, Hora, & Valente, 2017). API changes may be classified into *breaking changes* and *non-breaking changes* (Dig & Johnson, 2006). Breaking changes are those that break backward compatibility due to either removal or modification of API elements, resulting in consumers potentially facing compilation errors after updating. Non-breaking changes are changes that preserve compatibility, typically only involving addition of new functionalities. Because of this, migration between API versions which only include non-breaking changes does not cause negative effects to consuming applications (Xavier et al., 2017). Hence, API providers typically aim to only make non-breaking changes to their API, by designing their API for extensibility from the very start, so that there is loose coupling between versions (Medjaoui et al., 2018). In the event of breaking changes however, unless there is a high return-on-investment, developers will often not voluntarily migrate to a newer version (Espinha, Zaidman, & Gross, 2015). However, when integrating with a web API, developers can often not afford to do so since the API provider sets the pace for migrating to newer versions (sometimes eventually removing older ones altogether), forcing consuming developers to migrate. If this is the case, it should be ensured that migrating to a newer version of the API may be done smoothly, which is why managing multiple versions of an API to support existing consumers is an important capability the API provider should offer. Another helpful tool in ensuring consumers are able to adapt to changes made to the API, is adequately notifying consumers of any planned changes to the API. In order to do so, the API provider should have a mechanism in place with which consumers may be notified of any API updates or possible downtime (De, 2017). This may be done, for example, through email, social media, release notifications or notifications within the developer portal.

### 3.5.2 Developer Enablement for APIs

An API program is much more likely to be successful when the developer community is actively involved. Hence, managing communities of internal, partner, and external developers is a fundamental aspect of API management, ensuring that steady engagement with developers is maintained (Vijayakumar, 2018; L. Weir, 2019). Oftentimes, developers like to know the views of other developers in the community

and may want to collaborate and share their API usage learnings and experiences with one another (De, 2017). Communication channels such as the developer portal, blogs and forums form a major part of collaboration and community management (Indrasiri & Siriwardena, 2018). Through blog posts and community forums, developers may share their experiences with API usage as well as best practices and advice. Additionally, these channels may also be utilized by developers to report any issues with an API or its usage to the API provider's support team.

#### Developer Onboarding

In order to start consuming an API, developers typically must first register with the API provider to get access credentials (De, 2017). This process, which is often referred to as *developer onboarding*, involves a self-registration process which may often be done within the developer portal (L. Weir, 2019). After having registered, developers may then view and select available APIs through the usage of a mechanism commonly referred to as *API discovery* (Patni, 2017). This involves browsing through a discoverable catalog of APIs, which is published and maintained by the API provider. This API catalog is often also referred to as an API registry or directory, and should be able to be searched based on various metadata and tags (De, 2017; Vijayakumar, 2018; L. Weir, 2019). After having selected an API, developers may register their apps to use it. Doing so typically prompts an *API key* being generated. The API key is often also referred to as an app key or client ID and is used to identify the app an API is registered to (De, 2017). Overall, the utilization of mechanisms such as API discovery and the API catalog ensures that the developer onboarding process is made as easy as possible, improving the developer experience as a result.

#### Documentation

Another way with which API providers may foster and assist their developer community, is through providing API documentation. Comprehensive and rich documentation is considered to be empirical for the success of an API (L. Weir, 2019). By reviewing an API's documentation, consumers are able to review information regarding the API, such as the used standards, version, URI syntax, constraints, available resources, error codes and FAQs (Medjaoui et al., 2018; Vijayakumar, 2018; L. Weir, 2019). Additionally, documentation pages may also include start-up documentation to help developers quickly get started in using the API, as well as tutorials and sample code (De, 2017). Furthermore, documentation often includes a *Service-Level Agreement (SLA)*. This agreement defines the API's non-functional requirements, which may include matters such as the expected throughput, response time, and maintenance or downtime information. Considering that keeping API documentation synchronized with the implementation of the API itself is often a difficult task, documentation standards and frameworks such as RAML, Swagger and API Blueprint are often used. Many of these frameworks provide API providers with tools to automatically generate documentation from the API's interface (De, 2017). API providers typically provide documentation for their API on platforms such as the developer portal or the API catalog (Medjaoui et al., 2018). Overall, quality and user-friendly API documentation is key to its successful adoption, which is achieved by supplying consumers with vast amounts of information, bridging communication gaps as a result.

### Developer Support

In addition to providing developer support through the usage of community forums and documentation, support can be provided by allowing developers to test API behavior before actually deploying it. This may be accomplished in a number of ways. First, developers may be provided with an embedded *test console* within the developer portal. This console allows developers to freely play around with an API and test out behavior in an interactive manner without having to write any code (De, 2017; Patni, 2017). Another similar tool developers may be provided with is called an *API sandbox*. This tool allows developers to make calls against a simulated version of an API without any consequences, in a separate and virtual production environment (Medjaoui et al., 2018). Additionally, sample code may be provided, demonstrating the use of APIs and acting as a quick start guide (De, 2017).

## 3.5.3 Secure, Reliable and Flexible Communications

As described in Chapter 3.4, the main component that forms the heart of any API management platform is called the API Gateway. Its main functions, which are depicted in Figure 3.6 below, constitute into enabling secure, flexible, and reliable communication between the back-end services and apps (De, 2017). The various, most common capabilities and features that are typically offered by API gateways are described in further detail below.

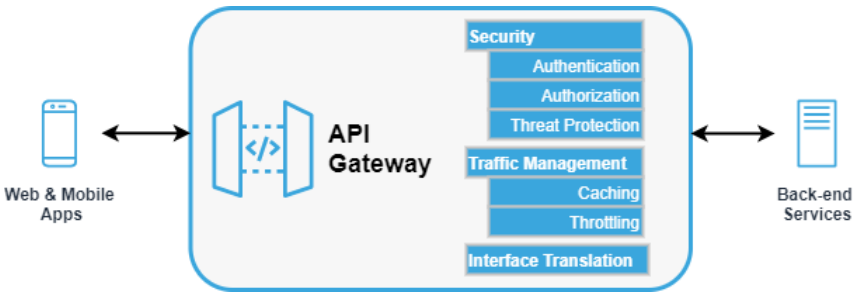



FIGURE 3.6: Common capabilities, as well as the features they consist of, as provided by the API Gateway.

### Security

Considering that APIs provide access to valuable and protected data and assets, it is paramount for APIs to be well-secured so that underlying assets are protected from unauthenticated and unauthorized access (De, 2017). One method which may be used to secure an API is the usage of an authentication method. *Authentication* is the process of uniquely determining and validating the identity of a client (De, 2017). One method to authenticate clients is the usage of API keys, which are simple identifiers of a caller (Vijayakumar, 2018). The API key is often also referred to as an app key or client ID and are commonly issued via self-service capabilities in the API developer portal for registered consuming applications (L. Weir, 2019). Another simple authentication method is called HTTP authentication, where credentials are provided by the caller in the HTTP header and validated against a Lightweight Directory Access Protocol (LDAP) server. Other more complex authentication methods

exist, which are based on issuing client certificates or tokens as opposed to a username and password (L. Weir, 2019).

A second method that may be used to secure an API is the usage of authorization policies. *Authorization* refers to having the ability to verify whether a caller has adequate rights to access a given resources, preceding authentication (L. Weir, 2019). One of the most commonly used authorization protocols is the OAuth 2.0 protocol. OAuth 2.0 is a standard for delegating authorization for accessing resources via HTTP (Patni, 2017). In order to do so, an access token which represents access rights for a subset of data for a short period of time, is handed over to the application making a call. When this access token expires, the calling application may automatically refresh it by presenting a refresh token, which may be exchanged to get a new access token with a renewed validity period (De, 2017). Authorization protocols are often used in combination with encryption to ensure that the communication channel through which they are sent is secured. Encryption can be performed by using the SSL or TLS protocols.

Another important security aspect is ensuring protection against threats and attacks, considering that APIs open valuable data and assets outside the firewalls of an organization (De, 2017). This vulnerability may result in hackers trying to attack back-end systems by overloading them with high amounts of traffic through Denial-of-Service (DoS) attacks. Such attacks could be protected against by, for example, implementing a firewall in front of the API gateway as a first line of defence, detecting and protecting against malicious threats (L. Weir, 2019). Another technique which may be used to protect from DoS attacks is the spike arrest mechanism, which identifies an unexpected rise in API traffic and drops traffic exceeding the pre-imposed limit. For public APIs, the likelihood of bad actors attempting attacks using malicious content is high, but they can also occur to private and partner APIs (De, 2017). Content-based attacks may take the form of malformed XML or JSON, malicious scripts or SQL being sent as part of the message payload. Hence, malformed request formats or malicious content within the payload should be able to be detected, identified and protected against.

#### Traffic Management

Considering APIs are highly available services, they risk being flooded with large loads of traffic and may crash or deliver poor performance as a result. Hence, they must be safeguarded from excessive incoming traffic (Jacobson et al., 2011). This practice is commonly referred to as *traffic management*. There are both business and technical reasons for controlling the amount of traffic that an API handles. Business-level traffic management is concerned with offering a different business value to different classes of customers, since each customer class may be willing to pay differently for access to the API (De, 2017). A common method through which this may be achieved is the implementation of consumption quota. Consumption quota define the number of API calls that an app is allowed to make to the back-end over a given time interval, which is typically expressed in the number of API calls that can be accepted over the course of an hour, day, week or month (De, 2017; Jacobson et al., 2011). Calls exceeding the quota limit may be throttled or halted, with the limit depending on what customer class the caller is assigned to. These customer classes may depend on the type of subscription (e.g. free or paid) or the type of business relationship the caller has in relation to the API provider. Depending on these classes,

the API provider may decide to prioritize traffic by processing calls made by customers of a higher class. In the event where the consumption quota is exceeded, the usage throttling mechanism may be utilized instead of fully halting calls made. When triggered, this technique results in slowing down API calls and can help to improve the overall performance and reduce impacts during peak hours (De, 2017).

As mentioned earlier, there also exist technical motives for managing traffic. This traffic management perspective is called operational traffic management, which focuses on maximizing API uptime, performance and efficiently routing traffic. One method with which this may be achieved is through the usage of caching. Caching is a mechanism used to optimize performance by responding to requests with static responses that are stored in-memory (De, 2017). When apps make requests on the same URI, the cached response can be used to respond instead of forwarding those requests to the back-end server, thus improving reponsiveness and performance (De, 2017; Vijayakumar, 2018). Another method with which incoming traffic may be managed, is the usage of load balancing. Load balancing helps to distribute API traffic to the back-end services, by routing it to the appropriate resource that is hosting the service. Other additional traffic routing capabilities may be required, such as service dispatching. This allows for selecting and mediating the correct back-end service based on the specific resource the call is targeting (De, 2017; L. Weir, 2019). In some cases, the API gateway may need to coordinate multiple calls to other services in order to deliver a pre-defined process (L. Weir, 2019). Doing so helps in improving the overall performance of the client by reducing latency introduced due to multiple API calls (De, 2017).

#### Interface Translation

When creating an API with the goal of exposing data and services, API providers need to ensure that the API interface is intuitive for developers to use. However, the interface for the API is likely to differ from that of the back-end services that it exposes (De, 2017). Because of this, the API gateway should be able to transform the API's interface to a form that the back-end can understand. In terms of formats, the back-end might expect message payloads in SOAP or XML, which cannot be easily consumer by the API consumer. Hence, these payloads should be converted from one format to another, for example, from XML/SOAP to JSON (L. Weir, 2019). Additionally, most back-end systems host services also provide a SOAP interface for consumers, even though SOAP is not a protocol that is suitable for APIs to build apps with (De, 2017). Hence, protocol transformation from SOAP to a more popular protocol such as REST should be supported.

### 3.5.4 API Auditing, Logging and Analytics

Organizations need to have insight into their API program to justify and make the right investments by understanding how their API is used, know who is using it, and what value is being generated from it (De, 2017). When exposing an API as an API provider, it is possible to collect a wide variety of operational and business data through the API gateway. This data may be analyzed to guide future decisions and answer a variety of questions. These include matters such as determining whether interest in the API is increasing among the developer community, how many apps utilize the API's services and how well the API is performing in terms of response

time and throughout (De, 2017). In order to formulate answers to these considerations, analytics and monitoring should be performed, which are critical activities from both the technical and business perspectives (Indrasiri & Siriwardena, 2018). Various types of analytics and monitoring capabilities exist, which each have different uses and are described below.

#### Activity Logging

Activity logging provides API providers with basic logging of access, consumption, performance, and any exceptions in relation to their API (De, 2017). In order to do so, information should be captured on the identity and location of callers by logging client IP addresses, as well as the date and time when a request was received and the response was sent (De, 2017). Additionally, performance metrics should be captured and logged. These metrics may include error rates, types of errors, system performance, latencies in request handling, and timeouts (Jacobson et al., 2011). These metrics should be utilized to create a proper and holistic understanding of how APIs are being used, by whom, when, and from where (L. Weir, 2019). Furthermore, these metrics can be used to define alerts and notifications to ensure that SLA agreements, such as availability, throughput, and response times, are not being breached (Jacobson et al., 2011; L. Weir, 2019). Additionally, metrics such as the number of unique users, the number of registered developers, and the number of apps built using the APIs, may be used to generate business value reports. These reports gauge the monetary value associated with the API program and the revenue generated from it (De, 2017). While some APIs may be directly monetized, many use an indirect model for monetization instead.

#### Health Monitoring

Health monitoring, or service monitoring, is performed to ensure that the gateway is up and running, as well as gaining real-time and historical-based insights about individual services, their interactions with other services, and their overall performance (Gadge & Kotwani, n.d.; L. Weir, 2019). This type of monitoring assists in tracing API traffic and is typically carried out during day-to-day operations and when debugging, identifying bottlenecks or troubleshooting services (Indrasiri & Siriwardena, 2018; L. Weir, 2019). Additionally, statistics that track the latency for back-end calls can be collected. These metrics may then be used as input to provide reports on errors raised during the processing of incoming API traffic, or ones that are received from the back-end (De, 2017).

## 3.6 Actors & Roles in API Management

Now that the API management discipline and the various activities and capabilities it comprises have been elaborated upon, the various actors, roles and positions that exist in the domain of API management are presented. Across literature on the topic, four authors mention API teams, roles or actors as part of their work. The first of which, Jacobson et al. (2011), mention 7 different roles, alongside a set of desired qualifications and the tasks whom they are responsible for. Alternatively, L. Weir (2019) mentions 11 roles, which are divided among two teams: API product teams and platform teams. According to the author, roles that are assigned to the former team are responsible for the individual APIs, and are therefore also responsible for executing all of the activities involved in the API life cycle. Roles assigned to the

latter team are responsible for the underlying platform on top of which the APIs run. A third author, De (2017), discusses various roles, teams and committees in relation to the API governance approach, which is discussed in further detail in the next subsection of this chapter. In De's work, 12 distinct roles are mentioned, which range from business oriented roles, to technical roles, and roles related to development methodologies such as SCRUM and DevOps. Additionally, the 'governance committee', as well as 5 different teams are mentioned. However, it is not made clear what roles individual roles are assigned to these teams. The last of these four authors that mention roles or actors involved in API management is Medjaoui et al. (2018). In his work, this author 11 distinct roles, which are categorized into two different categories: business roles and technical roles.

In comparison to the work of other authors, it was observed that the roles mentioned as part of the work of Medjaoui et al. (2018) most accurately matches the roles mentioned across literature. Additionally, the proposed categorization serves as a suitable foundation to discuss and summarize the individual roles and actors that were identified. As such, this structure was adhered to and complemented and supplemented with descriptions and roles that match the aforementioned categorization. The resulting roles are listed in Table 3.2 and Table 3.3 below, and are divided among two categories. The first roles category are called business roles. The individuals who take on these roles are primarily focused on the business side of the APIs (Medjaoui et al., 2018). Oftentimes, they are charged with the responsibility of speaking in the customer's voice, and aligning the API with clear strategic goals. The second set of roles consist of technical roles. These roles are focused on the technical details of implementing the API's design, testing, deploying, and maintaining it in a healthy, usable state throughout its active life (Medjaoui et al., 2018). Ordinarily, these roles are responsible for representing the voice of the IT department, including advocating for secure and scalable implementations that can be properly maintained over time.

TABLE 3.2: API Business Roles.

| Role | Description |
|---|---|
| API Product Owner/Manager | The product owner or manager is the main point of contact for the API. They are responsible for ensuring that the API has clear Objectives and Key Results (OKRs) and Key Performance Indicators (KPIs), monitoring the API and successfully guiding it through the full API lifecycle. Additionally, this role is responsible for defining and describing what work needs to be carried out on the API by other teams and to ensure the expected developer experience (design, onboarding and ongoing relationships) meet the needs of the API consumers (De, 2017; Jacobson et al., 2011; Medjaoui et al., 2018; L. Weir, 2019). |
| API Designer | The API designer is responsible for all aspects of the design. This includes making sure the physical interface is functional, usable, and offers a positive experience for developers. The designer also needs to make sure the API helps the team to achieve the identified business OKRs. Oftentimes, the designer is the first line of contact for API consumers and may be responsible for taking on the "voice of the consumer" when helping the team make decisions about the look and feel of the API. Finally, the designer may be called upon to make sure the overall design matches established organization-wide style guidelines (Medjaoui et al., 2018). |
| API Technical Writer | The API technical writer is responsible for writing the API documentation for all stakeholders connected with the API product. This includes not just the API consumers, but also the internal team members as well as other stakeholders from the business community. As a result, technical writers often work closely with the API designer and product owner to make sure the documentation is accurate, up to date, and in keeping with the organization's design and style guidelines (Medjaoui et al., 2018; L. Weir, 2019). |

Continued on next page

Table 3.2 - continued

| Role | Descriptions |
|---|---|
| API Evangelist | The API evangelist is responsible for promoting and supporting the API practice and culture within the organization, ensuring that all internal developers using the API understand it and can accomplish their goals with it. Evangelists are also responsible for listening to API consumers and passing their feedback on to the rest of the product team. Evangelists may also be responsible for creating samples, demos, training materials, and other support activities in order to maximize the developer experience (Jacobson et al., 2011; Medjaoui et al., 2018; L. Weir, 2019). |
| Developer Relations | The developer relations role, sometimes called the developer advocate or DevRel role, is usually focused on external use of the API. Like the API evangelists, DevRel staff are responsible for creating samples, demos, training materials and listening to API consumers and helping turn their feedback into fixes or features for the API team. However, unlike internal evangelists, DevRels are also often tasked with "selling" the API product to a wider audience, and as such may participate in customer onsites, presales activities, and ongoing product support for key customers. Additional duties may include speaking at public events, writing blog posts or articles on how to use the product, as well as other brand-awareness activities in order to help the team reach their stated business goals (Medjaoui et al., 2018; L. Weir, 2019). |
| Business Analyst | The business analyst is a functional expert in the business domain that the API relates to. Although, the product owner should be the domain expert, in practice, they may require additional support. Architects, developers, and technical writers may need additional functional support, and this role can therefore offload some of these activities from the product owner so that they can focus on other value-adding activities. Furthermore, the business analyst gathers the business requirements for API enablement and identifies the services to be exposed as APIs (De, 2017; L. Weir, 2019). |

TABLE 3.3: API Technical Roles.

| Role | Description |
|---|---|
| Lead API Engineer | The lead API engineer is the key point of contact for all the work related to the development, testing, monitoring, and deployment of the API product. The API lead engineer is responsible for the technical details regarding building, deploying, and maintaining the API. The lead engineer is the one with the responsibility to coordinate the other technical members of the team (De, 2017; Medjaoui et al., 2018). |
| API Architect | The API architect is responsible for the technical architectural design details for the API itself, as well as ensuring that the API can easily interact with required system resources, including APIs from other teams. It is the responsibility of the API architect to advocate for the overall software and system architecture of the entire organization. This includes supporting security considerations, stability and reliability metrics, protocol and format selections, and other nonfunctional elements that have been established for the organization's software systems (De, 2017; Medjaoui et al., 2018; L. Weir, 2019). |
| Front-end Developer | The frontend API developer (FE) is responsible for ensuring that the API offers a quality consumer experience. This entails helping to implement the company's API catalog, developer portal, and any other activities related to the frontend or consumer end of the API. Similarly to the designer role on the business side, the FE advocates for API consumers, but from a technical point of view (De, 2017; Medjaoui et al., 2018; L. Weir, 2019). |
| Back-end Developer | The back-end developer (BE) is responsible for the details of implementing the actual interface of the API, connecting it to any other services it needs to complete its work, and faithfully executing on the vision of the PM and API designer's description of API's functionality and technical implementation. It is the responsibility of the BE to ensure that the API is reliable, stable, and consistent once it is placed into production (De, 2017; Medjaoui et al., 2018; L. Weir, 2019). |
| Test/QA Engineer | The API test/QA (quality assurance) engineer is responsible for everything related to validating the API design and testing its functionality, safety, and stability. Typically, the test/QA role is charged with writing (or helping the FE/BE write) the actual tests and making sure they run effectively and efficiently. Oftentimes, these tests go beyond simple bench tests and behavior testing and includes ensuring that there are tests in place for interoperability, scalability, security, and capacity (De, 2017; Jacobson et al., 2011; Medjaoui et al., 2018). |

Continued on next page

Table 3.3 - continued

| Role | Descriptions |
|---|---|
| DevOps Engineer | The DevOps role is responsible for every aspect of the building and deployment of the API. This includes monitoring the API's performance to make sure it is in line with the stated technical KPIs and is properly contributing to the business-level OKRs. Typically, this entails work on the delivery pipeline tooling, managing the versioning schedule and supporting any rollbacks of broken versions, if needed. The DevOps role is also responsible for maintaining a dashboard showing real-time monitoring data as well as aiding in the review, diagnosis, and repair of any problems identified while the API is in production (De, 2017; Medjaoui et al., 2018). |

## 3.7 API Governance

Now that the topic of API management has been described, including the most common activities and capabilities it consists of, as well as the roles and actors involved, the API governance approach may be introduced. API governance provides a policy-driven approach that helps to enforce standards and checkpoints throughout the API lifecycle, which has been elaborated upon in Section 3.5.1. The policies driving governance are highly influenced by business objectives and goals, and may broadly be categorized into design-time governance and runtime governance policies (Gadge & Kotwani, n.d.). Design-time governance policies includes matters such as defining standards for API definitions, ensuring conformance to RESTful API design guidelines, as well as guidelines and standards for interface documentation, development and testing (De, 2017; Gadge & Kotwani, n.d.). Runtime governance is mainly concerned with tracking the life cycle of an API through policies related to routing, throttling, load balancing, rate limiting, as well as guidelines and policies on monitoring, deployment and versioning strategies (De, 2017; Gadge & Kotwani, n.d.). Generally speaking, standards and principles defined as part of API governance provide API quality assurance, such as security, availability, scalability, and reliability (De, 2017). Several phases can be discerned in relation to API governance, encompassing activities, starting with the API proposal all the way to its adoption, through requirements gathering, build and deploy, and operations during general availability. A comprehensive framework describing the various phases the API governance approach consists of is presented by De (2017), consisting of the following:

1. *API proposal*: During this first phase of API governance, a request for designing a new API or requests towards changing an existing API are proposed by the organization. There may be several reasons for doing so, including new business agreements, changes in existing business agreements, or a new change request being submitted to the API governance body.

2. *Technical requirements gathering*: Once the API proposal has been submitted, the technical requirements are gathered so that the exact specifications for the API may be created. In order to do so, API architects and business analysts work together to compose API definitions. First however, several decisions must be made, such as which API specification standard will be used and which versioning approach will be adhered to.

3. *Build and validate*: After the API specification has been composed and all requirements have been gathered, the development process commences. During this phase, test scripts are created and APIs are validated for compliance with

the composed API specification. Additionally, several design-related decisions have to be made, such as what best practices and tools will be used during development, and which testing approach should be utilized.

4. *General availability*: Once the API has been implemented and deployed, it needs to be published to a platform such as the developer portal so that it may be consumed. First however, commercial questions have to be answered, such as which monetization model should be used. Additionally, considering that the APIs may expose sensitive data to consumers, the terms and conditions for the use of the APIs and the associated data as part of the SLA should be composed by a legal team. This also includes defining the necessary steps that should be taken in case of any SLA violations. After having been approved, the API may be deployed and released for general availability. Once deployed, monitoring has to be implemented in order to track performance and usage metrics.

5. *Adoption*: During this last phase of API governance, the API is adopted among consumers whom start integrating it into their applications. In order to facilitate this, easy but secure signup and onboarding should be ensured for consumers and their apps. Furthermore, monitoring should be utilized in order to gain insights on the manner in which the API is used by consumers. Lastly, strategies detailing the course of action that should be taken when new versions of the API are introduced should be formulated, so that consumers and their applications are not negatively impacted.

## 3.8 Background

In this section, all of the existing frameworks, tools, models, reports and case studies that the authors of this work were able to identify and that are related to the scope and goal of this study are summarized and evaluated. These artefacts are classified into four categories, based on their characteristics and use cases. The first of these categories consists of API management maturity assessment frameworks, which bear the closest resemblance to the artefact this study seeks to design. The second category comprises artefacts that are aimed towards evaluating and comparing API management platforms or gateways with one another. Next, an advisory report is presented, which is targeted towards assisting organizations with implementing API management processes. The last category consists of artefacts that are not publicly available, and are oftentimes produced on a commercial basis. Lastly, this section concludes with a summarizing comparison matrix in which all discussed artefacts are compared to one another, as well as to the artefact this study intends to compose.

### 3.8.1 API Management Maturity Assessment Frameworks

In this section, two API management maturity assessment frameworks are discussed. The goal of these maturity frameworks is to assist API providers with assessing the degree of maturity corresponding to their API management-related business processes. These processes are divided into several categories and maturity levels, so that API providers may determine their degree of maturity depending on whether they have implemented them.

**Accenture - API Management Suite API Maturity Model**

Based on their experience with implementing API programs, Accenture Technology Labs has developed an API Maturity Model (Tung, 2014). This model consists of 5 maturity levels, and is aimed towards helping organizations identify the maturity stages their API is located in. The Accenture API Management Suite API Maturity Model, authored by Tung (2014), was originally published in 2013, after which it was improved and re-published in 2014. The latter Maturity Model is discussed in this subsection, and may be utilized to explore the capabilities an organization requires in order to unlock each stage to enable greater efficiency, agility, and scale. The model comprises five stages, or maturity levels: 0. *Ad-hoc*; 1. *Organize*; 2. *Tactical*; 3. *Critical*; 4. *Industrial*. These maturity levels are mapped onto five distinct dimensions, which detail which processes an organization should implement in their journey from API enablement to industrialization. In short, these dimensions comprise the following:

1. *Strategy and Governance*: the API strategy determines the business objectives, which includes ensuring developers operate in an efficient fashion. By supporting strategy, governance is applied, resulting in a consistent approach for defining, developing, publishing, supporting and deprecating APIs in a manner that is well-structured and reliable for both API producers and consumers.

2. *Architecture*: at its core, the architecture supports the terms of service for the API, whether a program guarantees any SLAs, bundles all users in a single category, or offers distinct SLAs that may vary across apps, users, geographies, and requests. This is accomplished through the implementation of capabilities such as effective traffic and access management, as well as monitoring and monetization.

3. *Development Process*: the development processes of an API focuses on combining and transforming existing back-end services to make them manageable for the consuming developers, and requires its own standardized processes and tools for development, testing, deployment, and promotion. In order for an organization to increase its maturity in this dimension, it is suggested to automate and standardize processes regarding testing and security.

4. *Developer Community* : Tung (2014) argues that developers are key actors towards realizing an API's success, as well as increased efficiency, and promoting innovation and direct monetization. Hence, organizations should foster their developer community through collaboration and community management.

5. *Optimization*: Analytics play a critical role in ensuring the success of APIs, by providing necessary insights which range from assessing value, to identifying how to best improve performance. In order to achieve industrialized API analytics, organizations must go beyond traditional trend reporting, and instead focus on assessing real-time and predictive analytics.

Even though this maturity model fails to address certain core API management-related processes and capabilities such as versioning, threat protection and lifecycle management, it seems to be relatively complete in its entirety. However, its incompleteness might be caused due to the fact that this model is relatively out-dated, and has since been deprecated, judging from the observation that it is no longer available on Accenture's official website. Additionally, in part due to its industrial foundation and commercial nature, it is unclear as to how the contents of this model have

been populated. Despite this however, it provides organizations with useful and platform-independent advice on how to increase their degree of maturity regarding API management.

#### Endjin - API Maturity Matrix

Another framework that bears resemblance to the API management maturity assessment tool or framework this study seeks to compose, is Endjin's API Maturity Matrix (Endjin, 2017). This maturity matrix is a tool which was developed to aid business decision makers in assessing their organization's ability to evolve towards an API driven business model. Doing so is accomplished by assessing the organization's degree of maturity (low, medium or high) in relation to a set of categories. This assessment is performed by having the organization fill out their perceived degree of maturity related to each category, based on which practical suggestions for improvement are then provided.

While this maturity matrix comprises a selection of categories that are relevant in the scope of API management such as governance, documentation and support, it mainly focuses on strategies and commercial aspects. As a result, many API management-related aspects such as traffic management and community management are missing from the matrix, especially when, for instance, compared to the earlier described maturity model by Accenture. However, it does provide organizations with clear suggestions for maturity improvement, but these suggestions rely on the organization first making a self-assessment, which may be difficult for them to do.

### 3.8.2 API Gateway & Management Platform Comparisons

This second category comprises artefacts that are aimed towards evaluating and comparing API management platforms or gateways with one another. First, a framework that consists of a whitepaper presenting various API management platform requirements, paired with an evaluation matrix which details a set of evaluation criteria that may be used to evaluate management platform vendors, is discussed. Next, an academic paper which aims to compare API management features offered by various API gateway providers with one another is discussed. Lastly, a whitepaper describing, among other things, essential API management platform capabilities is dissected.

#### WSO2 - API Management Platform Technical Evaluation Framework

In 2015, WSO2 has published a whitepaper that describes digital business goals, outlines API oriented IT initiatives, and presents API management platform requirement categories (WSO2, 2015). Alongside this whitepaper, an evaluation matrix spreadsheet is presented that details a set of evaluation criteria, which may be used to evaluate API management platform vendors (Haddad, 2015). In illustrating the discipline of API management, two types of APIs are discerned: naked and managed APIs. A naked API is considered to be not monitored, managed, secured, documented or accessible through a self-service subscription portal. On the other hand, a managed API is thought to be actively advertised and subscribe-able, available alongside a published SLA, secured, authenticated, authorized, protected, as well as being monitored and monetized using analytics. WSO2 (2015) goes on to

argue that to move from naked to managed APIs, the *API façade pattern* should be implemented, which enables teams to layer network-addressable endpoints, monitor usage, enforce usage limits, manage traffic, and authorize consumers. According to WSO2, an API management infrastructure should guide teams towards best practices with regards to six main focus areas, which are summarized in short below.

1. *API design and implementation*: APIs differ from back-end services by design, and expose a RESTful and resource oriented facade that enforces security and service policies. WSO2 (2015) states that the main activity categories API design and implementation is composed of include; (1) *API design*; (2) *API façade development*; (3) *Service level definition*; (4) *API Mediation*; (4) *API documentation*; (5) *API testing*.

2. *API Security*: According to WSO2 (2015), security is of paramount importance. Oftentimes, API management platform security capabilities are shared with gateways, firewalls, identity management, and access control components. WSO2 states that the main activity categories that should be offered by API management platforms are; (1) *Access control, authentication, and key management*; (2) *Entitlement assertions*; (3) *Attack prevention*; (4) *Confidentiality, integrity, and privacy*; (5) *Identity management*; (6) *User management*.

3. *Publication and community engagement*: WSO2 (2015) argues that by incorporating a collaboration space that encourages community participation and community management, API management platforms actively engage internal and external developers. This is accomplished by utilizing this collaboration space to establish a feedback channel between API consumers and providers. According to WSO2, API management platforms should increase API adoption by; (1) *Streamlining API publication and following DevOps best practices*; (2) *Engaging API consumers and lowering API adoption barriers*; (3) *Actively managing API developer communities*; (4) *Fostering a thriving, business-oriented API economy*; (4) *Identity management*; (5) *Streamlining API DevOps publication tasks by integrating API lifecycle activities with run-time infrastructure*.

4. *Monitoring and run-time management*: According to WSO2, API management platforms should incorporate monitoring and management capabilities, implement operational best practices, and enforce Quality of Service (QoS) policies. An advanced API management platform should support; (1) *Configuration management*; (2) *Release management*; (3) *Patch management*; (4) *Service level management*.

5. *API lifecycle, policy, and community governance*: Considering the fact that API governance is heavily influenced by IT business goals and objectives, leading API management platforms often provide analytics that support the assessment of IT business value (WSO2, 2015). To this end, an API management platform should provide for the following governance related capabilities; (1) *Meta-data management*; (2) *Service level management*; (3) *Version management*; (4) *Lifecycle management*; (5) *Usage management*; (6) *Portfolio management*.

6. *API analytics*: According to WSO2, adoption and usage metrics should be utilized to invest future development resources, plan API infrastructure scales, and rationalize the API portfolio. Furthermore, using an API analytics dashboard can aid teams in ascertaining development experience, validating security and service level compliance and quantifying API brand value.

As part of the evaluation matrix mentioned earlier, the six main focus areas described above and the respective activity categories they comprise, are further subdivided into hundreds of individual processes and questions. As such, as an API management assessment artefact, this matrix is considered to be very complete with regards to its contents. However, from examining the matrix, it is unclear as to whether organizations are supposed to answer these questions and subsequently assign themselves scores based on their answers, or whether they are assisted by WSO2 in this process. Furthermore, while a "weighted score" column is included in the matrix, it is unclear what these weights are based on, or what formula is used to calculate the weighted scores. Regardless, by answering these questions, organizations may decide which API management platform is best suited to cater to their needs. However, due to the fact that this whitepaper and matrix were written by WSO2, which is a commercial API management platform provider, organizations are steered towards selecting their platform. Additionally, regarding transparency, it is unclear as to how the contents and structure of the matrix have been composed.

#### Gamez - API Gateway Feature Comparison

As part of their work, which seeks to analyze the API Gateway paradigm and propose a SLA-Driven solution in an API Gateway design, Gámez Díaz et al. (2015) have compared API management features offered by various API gateway providers. This collection of providers consists of 13 Gateways including: *3Scale, Akana API Gateway, API Umbrella, Apiaxle, Apigee Edge, Axway API Gateway, Azure API Management, CA API Gateway, Mashape, Mashery API Gateway, Monarch API Manager, Repose* and *WSO2 API Management*. The aforementioned gateways were compared as based on a set of features, including; (1) *Security*: comprising basic authentication, advanced authentication and attacks protection; (2) *Pricing plans support*: consisting of free tier, requests limit and enterprise plans; (3) *Lifecycle Control*: composed of API versions and orchestration; (4) *Other features*: comprising open source, load balancing, cache, response conversion and documentation.

Confusingly, in their work, Gámez Díaz et al. (2015) use the terms 'API gateway' and 'platform' interchangeably. As a result, it is unclear whether the intention of the authors was to analyze features offered by the API gateway component, which is one of the main architectural components offered by the listed API management platform providers, or features provided by the platforms as a whole. Aside from the observation that the set of features the aforementioned API gateways are compared on is very limited when compared to work on API management by authors such as De (2017), discussion of the results presented in the matrix shown above is scarce. However, Gámez Díaz et al. (2015) state that almost all studied API gateway services offer the same core functionality, which is requesting authorization and quota establishment so that various billing plans may be created.

#### Broadcom - API Management Playbook

In order to promote their API management platform, called Layer7, Broadcom has employed CA Technologies to compose an 'API Management Playbook' (CA Technologies, 2019). This playbook is targeted towards helping its readers comprehend the various reasons for API's importance in business, the API lifecycle and its relation to API management, the essential capabilities of an API management solution, and the features offered by the Layer7 platform. After having highlighted a

set reasons for API's importance, as well as the types of members an API management team should be composed of, the API lifecycle is explained. According to CA Technologies (2019), the API lifecycle comprises six stages: *define, design, create, deploy, promote* and *initiate*. The API lifecycle is defined as *"an approach that governs the creation, deployment, promotion and optimization of APIs on the provisioning side, as well as their acquisition and usage by API consumers"*. Next, an evaluation method is presented which considers API management capabilities based on a collection of 13 use cases, or 'plays'. These use cases are broadly classified as having the goal of: API integration and creation, security, mobile and internet of things (IoT) development acceleration, and unlocking the value of data. Subsequently, these use case categories are then mapped onto eight main categories of API management capabilities. According to CA Technologies (2019), API management capabilities may best be categorized into categories that comprise topics such as API design, runtime, protection, access control, ownership, development, intelligence and discovery. These categories are then further sub-divided into a variety of capabilities.

While the presented collection of capabilities seems to be quite comprehensive, capabilities such as version management, auditing and logging seem to be missing. However, this playbook is effective in consulting organizations as to what API management solution is suitable in relation to their target use cases, which is the main purpose of this document. Nevertheless, the presented definition and structure of the API lifecycle, as well as the API management capability overview may be generalized in a broader context. It is clear however, that even though this work presents an useful overview of API management capabilities, this overview directly matches the features offered by the Layer7 solution. As such, it may be concluded that the main purpose of this work is to convince and attract potential customers to Broadcom's platform. Furthermore, due to the commercial nature of this document, it can not be considered transparent in the sense that the source of the presented information is not known.

### 3.8.3 API Management Advisory Reports

This artefact category comprises a consultancy report, which is aimed towards advising banks on how to implement API management practices.

#### Accenture - APIs: The Digital Glue

In addition to the maturity model described in Section 3.8.1, Accenture has published a consultancy report in 2019, advising banks on how to implement API management (Lees, Mallick, Paar, Fontenay, & Pimakova, 2019). Even though this report is specifically focused on the banking sector, and as a whole may thus be difficult to generalize and applied to other sectors, it contains several frameworks, figures and models that are related to this study. As such, relevant and interesting findings from this report are presented below. Lees et al. (2019) argue that effective API management for banks rests on four pillars: approach, technology, governance and ecosystem management. These pillars are summarized in short below.

1. *Approach*: According to Lees et al. (2019), a business-driven approach is the cornerstone of effective API management. In order to illustrate this, an API adoption approach is presented, which comprises 4 phases; (1) *Discover*; (2) *Build*; (3) *Run and Evolve*; (4) *Socialize and Publish*.

2. *Technology*: Lees et al. (2019) argue that appropriate technology decisions are essential to any effective API transformation program. A set of technological factors may be discerned, including; (1) *API Management Platforms*: these platforms are considered to be critical components of an API management structure, comprising a variety of features such as analytics, versioning and traffic management; (2) *API Catalog Management*: Discoverability is considered to be a key factor in allowing API reuse. This may be supported by a structured API catalog, which makes APIs discoverable; (3) *Monitoring*: in order to perform effective API management, Lees et al. (2019) state that a variety of monitoring capabilities are required in order to track elements such as usage statistics, and to identify whom is consuming APIs.

3. *Governance*: Lees et al. (2019) argue that strong governance and efficient processes aid towards creating a framework for effective API architecture delivery. To this end, a multi-tiered governance structure is proposed, consisting of three key teams; (1) *Central Teams*: teams that acts as a central design and quality authority for all APIs that are developed across the organization; (2) *Business Unit Design Teams*: teams that employ the guidelines and frameworks that were established by the central team, and submit their designs to central design authority, before commencing delivery. (3) *Delivery Teams*: small dedicated delivery teams, whom are supported by the business units and, based on common frameworks and standards defined by the central teams, are able to deliver the final API.

4. *Ecosystem management*: According to Lees et al. (2019), banks can no longer solely rely upon their internal resources and capabilities, but instead should collaborate and build networks. In order to do so, the following aspects should be implemented: (1) *Developer Portal*: oftentimes, the developer portal is the first line of communication for developers whom are interesting in using an organization's API; (2) *Facilitate Engagement*: among other things, facilitation of user or designer-led events such as hackathons and sandboxing of dummy APIs to test viability, may be utilized to increase innovation and provide access to new revenue streams; (3) *Monetization*: in order to create new revenue streams, organizations should work on developing appropriate monetization models, such as "freemium" access, tiered pricing and revenue sharing.

As mentioned earlier, this report is specifically focused on the banking sector, and as a whole may thus be difficult to generalize and apply to other sectors. However, the presented API management platform capability overview may be utilized by organizations seeking to implement API management processes, or may aid them in identifying API management platform capabilities that cater towards their needs. Furthermore, considering the aforementioned pillars are described in great detail and provide clear-cut guidelines for organizations to follow, this advisory report may be beneficial to organizations wishing to implement API management. However, when compared to the earlier described maturity models and frameworks, it may be difficult for organizations to self-assess their degree of maturity concerning API management.

### 3.8.4 Non-Publicly Available Evaluation Frameworks & Case Studies

In addition to the models, frameworks and reports described earlier throughout this chapter, a collection of frameworks, checklists and case studies exist that are not

publicly available. Oftentimes, these are produced on a commercial basis, requiring them to be purchased or require potential readers to sign up with the vendor, after which they are vetted based on a set of characteristics. Two of such frameworks and case studies were identified, which are described below.

#### Gartner - Guidance Framework for Evaluating API Management Solutions

The Guidance Framework for Evaluating API Management Solutions was published by Gartner in 2019 (Gartner, 2019). Similarly to those published by WSO2 and CA Technologies, this framework is aimed towards assisting technical professionals in selecting an appropriate API management platform. Unfortunately, considering the fact that this framework is exclusively commercially available, the authors of this study were not able to review it in its entirety. However, the general outline and structure of the framework is visible by reviewing the table of contents (Gartner, 2019). First, the framework uses the organization's intended API management platform use case and intended users as input. Next, commercially available API management platforms are evaluated on their offered capabilities and features concerning deployment and operations, secure, reliable and flexible Communications, developer enablement, API lifecycle management, and monitoring. These feature and capability categories are further sub-divided into a set of individual capabilities. In addition, readers of the framework are provided with a series of risks and pitfalls that should be avoided.

Interestingly, some of the capability category titles directly correspond to the wording used by De (2017) for the API management platform capability categories listed in Figure 3.4. For instance, part 2 of Gartner's framework, which is titled *Evaluate Features for Secure, Reliable and Flexible Communications* matches De's *Secure, Reliable and Flexible Communications* capability category. The same observation holds true when comparing De's *Developer Enablement for APIs* capability category with Gartner's *Evaluate Capabilities That Enable Developers*, which is a paraphrased variant of De's naming. Furthermore, a large portion of the individual capabilities Gartner's capability categories are composed of directly correspond to those of De (2017). For instance, Gartner's *Evaluate Features for Observability and Monitoring* category consists of the activity logging, user auditing, business value reporting, contract management and advanced analytics capabilities. This set of capabilities directly matches the ordering and the capabilities De's *API Auditing, Logging and Analytics* category consists of, excluding the service level monitoring capability. Judging from these observations, it may be concluded that this framework published by Gartner (2019) is grounded in literature, using De (2017)'s work on API management as a foundation.

#### Devoteam - API management at Liberty Global Inc

On their website, a Dutch company called Devoteam summarizes a case study in which the implementation of an API management platform at a large organization, "Liberty Global", is described (Devoteam, 2016). As part of this case study, the case organization is described to initially have an API gateway implemented. An API gateway is argued to be an incomplete solution when compared to an API management platform. Furthermore, the organization required API management capabilities such as developer and partner onboarding, lifecycle management, documentation and testing, and analytics, which Devoteam (2016) does not consider to be a

part of an API gateway.

In order to recommend an appropriate API management platform to the case company as based on their needs, an "integration cookbook" was created, documenting the principles and guidelines for the usage of the API management platform and the different integration patterns. Unfortunately, similarly to the earlier described framework by Gartner, this cookbook is not publicly accessible. However, Devoteam (2016) mentions that it comprises policies that range from security policies, such as OAuth 2.0, OpenID Connect, basic authentication and IP whitelisting, to operational policies and monitoring and auditing policies. Judging from this information, it seems as though this cookbook primarily focuses on strategy, governance, and API management vendor evaluation and comparison. Due to its commercial nature however, it is unknown as to how this cookbook was created, whether it is able to be used to assess an organization's degree of maturity in regarding API management, or how complete it is.

### 3.8.5 Comparison Matrix

In this section, a comparison matrix is presented in which all related work described throughout this chapter is compared to one another as well as the proposed artefact this study seeks to produce. The characteristics and criteria used in comparing these artefacts are largely based on the goal statement that guides this study, which was presented in Section 1.2, as well as the evaluation criteria presented in Table 3.4. These characteristics and comparison criteria are defined as follows:

- *Type*: the type of artefact that is described. This may include (focus area) maturity models, (evaluation) matrices or frameworks with which API management platforms are evaluated or compared, whitepapers, advisory reports, or case studies.
- *Missing contents*: the degree to which the API management artefacts contain all the (best) practices, capabilities and features API Management is comprised of. The assessment of this criterion is performed by using the activities and capabilities described in Chapter 3 as a benchmark.
- *Evaluation method*: the evaluation method utilized by the artefact. This may include maturity degrees as part of a maturity model, (weighted) scoring methods, or comparison of the capabilities offered by API management platforms or gateways.
- *Availability*: the availability of the artefact. The artefact may either be publicly available online, or commercially available by requiring it to be purchased or by requiring potential readers to sign up with the vendor, after which they are vetted based on a set of characteristics.
- *Grounded in*: the nature in which the contents of the artefact are grounded. This may include a foundation in academic literature, or industry-based knowledge and experience.
- *Published in*: the year in which the artefact has been published.
- *Transparent*: the degree of apparent transparency concerning the construction and design of the artefact, from the perspective of the reader or consumer of

the artefact. Aside from the degree to which the contents of the artefact are described and elaborated upon, an artefact is considered to be transparent when the standards, guidelines and design methodologies that were used in designing and constructing the artefact are described.

- *Platform-independent*: the degree to which the artefact is independent of any API management platform or gateway providers. An artefact is considered to be so, when presented overviews, models and figures as part of the artefact are able to be generalized across all possible API management platforms or gateways.

TABLE 3.4: Comparison of all related work and study's artefact, based on a set of characteristics and evaluation criteria.

| Framework | Type | Missing content | Evaluation Method | Availability | Grounded in | Published in | Transparent | Platform-independent |
|---|---|---|---|---|---|---|---|---|
| Accenture API Management Suite | Maturity Model | Versioning, threat protection, authorization | Maturity degrees | Public | Industry | 2014 | No | Yes |
| Endjin Maturity Matrix | Maturity Matrix | Lifecycle management, dev. onboarding, traffic management, interface translation, monitoring | Maturity degrees | Public | Industry | 2017 | No | Yes |
| WSO2 Platform Evaluation | Whitepaper & Evaluation matrix | Dev. support | Weighted scores | Public | Industry | 2015 | No | Yes [1] |
| Gámez Gateway Comparison | Comparison Matrix | Dev. onboarding & support, authorization, traffic management, monitoring | Capability comparison | Public | Literature | 2015 | Yes | Yes |
| Broadcom Playbook | Whitepaper | Version management, interface translation, logging | Capability comparison | Public | Industry | 2019 | No | Yes[1] |
| Accenture Advisory Report | Whitepaper | Dev. support, interface translation, logging | Capability comparison | Public | Industry | 2019 | No | Yes |
| Gartner Guidance Framework | Evaluation framework | Unknown [2] | Capability comparison | Commercial | Industry & Literature | 2019 | No | Yes |
| Devoteam Case Study | Case Study & Cookbook | Unknown [2] | Capability comparison | Commercial | Industry | 2016 | No | Yes |
| Proposed Framework or Tool | TBD [3] | None | TBD [3] | Public | Industry & Literature | 2021 | Yes | Yes |

[1] Published by commercial API management platform provider.
[2] Full contents unknown due to non-publicly available nature.
[3] To be determined and described in Chapter 4.

Conforming with this study's goal statement, the proposed API management maturity assessment framework seeks to incorporate all positive qualities from the related work discussed throughout this chapter. Furthermore, it aims to fulfill all the criteria imposed above. In order to determine which artefact type is best suited towards fulfilling this study's goal, available design science artefacts will be compared to one another. This process will be described in the next chapter of this thesis. Furthermore, it will be ensured that the contents of the tool or framework are saturated and complete, which is accomplished by the initial population of the framework through an extensive systematic literature review, as well as expert evaluation and validation through a case study. Doing so also ensures that the framework is

grounded in both literature and industry, as opposed to the artefacts discussed in this chapter. Furthermore, one of the most substantial value additions of the proposed framework as opposed to other artefacts is the degree of transparency with regards to its design and construction. With the exception of the work by Gámez Díaz et al. (2015), all discussed artefacts lack in transparency. This is mostly caused due to the fact that these artefacts were produced by companies that wish to safeguard the standards, guidelines and design methods that were used, so that they may maintain their competitive advantage. On the contrary, considering the fact that the artefact this study seeks to construct is supported by an extensive thesis describing every facet of its design, construction, evaluation and validation, full transparency is guaranteed for all its readers and users. Furthermore, this study's framework will be composed in such a way that it is API management platform-independent, publicly available and technology-agnostic, ensuring that the largest possible target audience may be reached. Lastly, it is ensured that the proposed framework or tool is recent, considering this work has been written in 2021. This is of added value due to the fact that the API management discipline is relatively new and rapidly evolving, and a significant portion of the related artefacts under evaluation have not been published within the past few years.

# Chapter 4

# Framework Design Tracing

In this chapter, the process of designing the API management assessment framework is traced and described. First, an appropriate and suitable design science artifact that aligns with this study's goal statement is selected. Then, the construction of the first version of the framework is elaborated upon.

## 4.1 Design Science Artifact Selection

In their work, Hevner and Chatterjee (2010) state that an artifact is a man-made object, created to solve a specific problem, as opposed to naturally occurring objects. Such an artifact must improve upon existing solutions to a problem, or otherwise provide a first solution to the identified problem. IT artifacts, which are the end product of any design science research project, are broadly categorized by Hevner and Chatterjee (2010), and March and Smith (1995) into four categories, as can be seen in Table 4.1 below.

TABLE 4.1: Design science artifact categories, as presented by Hevner and Chatterjee (2010), and March and Smith (1995).

| Category | Description |
|---|---|
| Constructs | Constructs or concepts form the vocabulary of a domain, and constitute a conceptualization that is used in describing problems within the domain and subsequently specifying solutions to these problems (March & Smith, 1995). Constructs form the specialized language and shared knowledge of a discipline or sub-discipline, and may differ in terms of their degree of formalization. |
| Models | Models are composed of sets of propositions or statements expressing relationships among constructs (March & Smith, 1995). By capturing these constructs and the relationships among them, a model may capture the structure of reality in order to form an abstraction or representation of it. |
| Methods | A method is a set of steps, in the form of an algorithm or guideline, which is used to perform a task (March & Smith, 1995). Methods are based on a set of underlying constructs, and a representation of the solution space, in the form of a model. |
| Instantiations | An instantiation is the realization of an artifact in its related environment (March & Smith, 1995). Instantiations operationalize constructs, models, and methods, and demonstrate their feasibility and effectiveness. |

As mentioned before, the discipline of API management consists of a wide variety of processes, activities, and capabilities, which need to be distilled into a tool or framework. In order to do so, it should consist of constructs so that the vocabulary of the domain of API management may be properly represented and conceptualized. In order to accommodate this, the usage of an artifact of the model or method type is appropriate. This hypothesis is based on the fact that methods are; (1) *systematic*: delivering rules on how to act, and instructions on how to solve problems; (2) *goal-oriented*: stipulating standards on how to proceed or act, to achieve a defined

goal; and (3) *repeatable*: inter-subjectively practicable (Braun, Wortmann, Hafner, & Winter, 2005; Mettler, 2011).

Models generally represent a formal description of aspects of the physical or social reality, for the purpose of understanding and communicating (Mylopoulos, 1992). According to Mettler (2011), based on the notion of the representation used, models may either be; (1) *descriptive*: giving an unprejudiced reproduction of some aspects of reality; (2) *explanatory*: delivering a depiction of causal connections to better understand reality; or (3) *predictive*: recommending an efficient solution state of a future reality. In short, models describe the state of reality through relationships among constructs, while methods focus on the specification of activities.

### 4.1.1 Maturity Models

There exist artifacts that, according to Mettler (2011), possess characteristics of both of the two artifact types described above. These artifacts are called *maturity models*, which show similarities to methods in the sense that they describe procedures which may be adhered to in order to reach a higher level of maturity. Moreover, maturity models are also regarded to match characteristics of the model artifact category, considering that they describe typical behaviour exhibited by a person or organisation at a number of predefined levels of maturity for each of several factors of the area under study.

Maturity models have been developed for organizations to use as an evaluative and comparative basis for improvement, in order to derive an informed approach for increasing the capability of a specific area within an organization (De Bruin, Rosemann, Freeze, & Kaulkarni, 2005). Moreover, maturity models have been designed to assess the maturity of a specific domain based on a set of criteria, and are a proven tool in the creation of collections of knowledge of practices and processes about a particular domain (Becker et al., 2009). In this context, maturity refers to being well equipped to fulfill a purpose, for example having a higher level of sophistication, capability, or availability of specific characteristics (Mettler, Rohner, & Winter, 2010). Maturity models consist of a sequence of maturity levels for a class of objects, which typically include organizations or processes (Becker et al., 2009). The aforementioned sequence represents an anticipated, desired, or typical evolution path of these objects as discrete stages. Maturity models aim to guide an organization along this path, starting from an initial state to maturity (Pöppelbuß & Röglinger, 2011). This initial state of maturity may be characterized by the organization having little capabilities with regards to the domain under investigation. On the contrary, organizations whose capabilities are of the highest maturity are located at the highest stage. In order for an organization to progress along the evolution path, criteria and characteristics relating to capabilities or process performance are included, which need to be fulfilled to reach a particular maturity level (Becker et al., 2009). To appraise an organization's maturity, maturity models are commonly applied to assess the as-is-situation regarding the given criteria, so that improvement measures may be derived and prioritized (Iversen, Nielsen, & Norbjerg, 1999).

As De Bruin et al. (2005) and van Steenbergen et al. (2013) argue, the most well-known maturity model within the field of Information Systems, is the Capability Maturity Model (CMM) from the Software Engineering Institute (Paulk, Curtis, Chrissis, & Weber, 1993). Studies have shown that since its inception, the CMM has inspired the development of hundreds of subsequent maturity models (De Bruin et al., 2005). These maturity models are aimed at a wide variety of domains and topics, including for example, project management (Crawford et al., 2007), corporate data

quality management (Hüner, Ofner, & Otto, 2009), service integration (Arsanjani & Holley, 2006), and offshore sourcing (Carmel & Agarwal, 2006). Another example, which has been discussed in Section 3.8, is the API management maturity model (Tung, 2014).

### 4.1.2 Focus Area Maturity Models (FAMM)

A specific type of maturity model is the Focus Area Maturity model (FAMM) (van Steenbergen et al., 2010, 2013), which is used to establish the maturity levels of an organization in a specific functional domain. This functional domain is described by the set of focus areas that constitute it (Jansen, 2020). Each focus area is composed out of a set of capabilities. van Steenbergen et al. (2013) define capabilities as *the ability to achieve a certain goal, making use of the available resources*. These capabilities are juxtaposed relative to one another in a maturity matrix, based on which a number of maturity levels may be discerned (van Steenbergen et al., 2013). In order to establish an organization's degree of maturity with regards to the functional domain, the organization is asked to answer assessment questions linked to the capabilities the maturity matrix consists of. Based on the results of this maturity assessment, the organization is then guided towards incremental development of the domain, through a set of improvement actions with regards to the (missing) capabilities (van Steenbergen et al., 2013).

In his work, Jansen (2020) describes the development of a focus area maturity model for the functional domain of software ecosystem governance. In doing so, a fourth component in addition to focus areas, capabilities and maturity levels is proposed: *practices*. This component may best be interpreted as the smallest building block the focus area maturity model consists of, which in the scope of software ecosystem governance, Jansen (2020) defines as *any practice that has the express goal to change the position of the platform in the software ecosystem*. Practices may depend on one another, requiring the dependent practice to be implemented before the other practice may be implemented. As a result of adding practices as an additional component, an adapted meta-model for the focus area maturity model, as originally introduced by van Steenbergen et al. (2013), is presented by Jansen (2020). This meta-model is depicted in Figure 4.1 below.

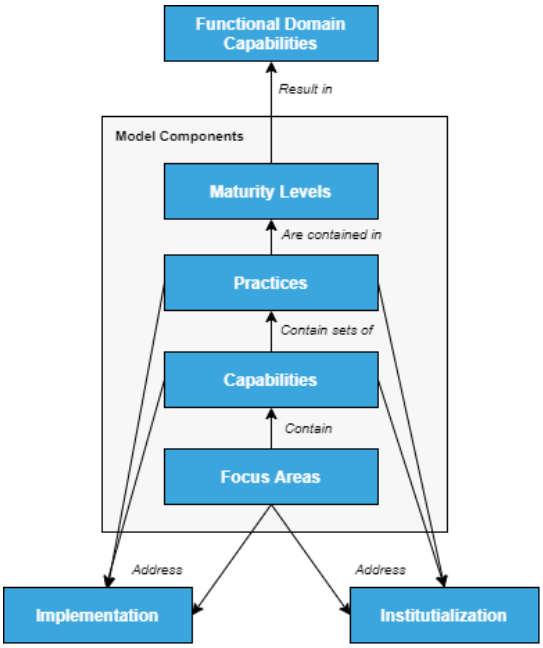



FIGURE 4.1: The adapted meta-model for focus area maturity models, as presented by Jansen (2020).

### Focus Area Maturity Model for API Management

As van Steenbergen et al. (2013) argue, when the goal of a study is to develop a functional domain step by step, the utilization of the focus area maturity model is more appropriate than 'traditional' maturity models, because it allows for a more fine-grained approach. This is caused due to the fact that most maturity models adopt the five maturity level structure defined by the CMM. The FAMM on the other hand, allows for an arbitrary amount of maturity levels to be distinguished. This characteristic is the first reason for deciding upon utilizing the FAMM as a foundation towards designing the API management assessment framework. As is evident from the literature review presented in Chapter 3 and Figure 3.4, the field of API management is very broad and extensive in terms of the (amount of) main- and sub-topics it comprises. Because of this, it is likely that these topics will comprise of a large collection of processes and activities, exceeding the standard five maturity level structure and thus calling for a non-fixed amount of maturity levels. This statement is further supported by the observation that in Accenture's maturity model (as described in Section 3.8), multiple capabilities are assigned to maturity levels.

Furthermore, FAMMs allow for dependencies between capabilities to be defined (van Steenbergen et al., 2013). There are many dependencies across API management; for example, many capabilities regarding lifecycle management depend on monitoring capabilities. Moreover, results from the previously conducted SLR (Mathijssen et al., 2020a) have shown that the field of API management consists of a plethora of smaller, individual activities and processes that are well-suited to be captured in the form of practices and capabilities, as defined by Jansen (2020). Due to this abundance of small, individual processes it is reasonable to assume that dependencies among them will occur, in addition to those existing between capabilities. As was mentioned in the previous subsection, Jansen (2020)'s adapted meta-model also allows for dependencies between practices to be defined, making it well-suited

to be utilized in structuring the functional domain of API management.

Moreover, the four capability categories shown in Figure 3.4 appear to be appropriate initial candidates to act as focus areas. As van Steenbergen et al. (2013) state, with regards to theoretical foundation, the fact that the focus area maturity model for a functional domain is assembled from its constituting focus areas, allows for better theoretical grounding. This characteristic satisfies this work's objective of ensuring that the API management assessment framework is grounded in theory and academics. Furthermore, the focus area maturity model is an appropriate artifact in closing the gap between theory and practice, and thus is of value to practitioners. This statement is supported by the FAMM's ability to represent information in a managerial way (e.g. providing an overview of the domain, quick to scan, easy to use), and is able to serve as a guideline or roadmap for practitioners to prioritize, and incrementally develop and improve capabilities in a functional domain (Spruit & Röling, 2014; van Steenbergen et al., 2013).

In conclusion, the decision is made to use Jansen (2020)'s adapted version of the meta-model for FAMMs, as originally introduced by van Steenbergen et al. (2013), to structure and design the API management assessment framework. This decision is made due to the need for flexibility in the amount of maturity levels that may be defined, as a consequence of the large amount of main- and sub-topics API management consists of. Additionally, Jansen (2020)'s meta-model allows for granular, individual processes and activities to be captured in the form of practices, between which dependencies may be defined. Lastly, FAMM's ability to provide practitioner with an overview of a domain, being easy to use and quick to scan, as well as allowing practitioners to incrementally develop and improve capabilities, allows for the existing gap between theory and practice in API management to be closed.

## 4.2 Constructing the API-m-FAMM

Now that the usage of the focus area maturity model as an artifact to capture the functional domain of API management has been motivated, the **API m**anagement **F**ocus **A**rea **M**aturity Model is introduced: the **API-m-FAMM**. Following the example of work conducted by Jansen (2020), a method for creating maturity models, which was introduced by De Bruin et al. (2005), is utilized in constructing the API-m-FAMM. As part of this method, six distinct development phases are discerned, as shown in Figure 4.2. The application of these phases to the API-m-FAMM is described in further detail in the following subsections and chapters.

### 4.2.1 Scope

The API-m-FAMM is targeted at the functional domain of API management. The SLR summarized in Section 1.3 as well as the literature and related work review described in Chapter 3 have shown that the API management domain is broad in terms of the capabilities, activities, and processes it encompasses. As an initial delimitation of the functional domain, the capability categories presented by De Bruin et al. (2005) in Figure 3.4 is adhered to. Considering that a prerequisite for performing API management as an activity, is for an organization to already have designed and created an API (as per the API lifecycle, depicted in Figure 3.5), the actual design and creation process of APIs itself is excluded from the API-m-FAMM. Furthermore, the

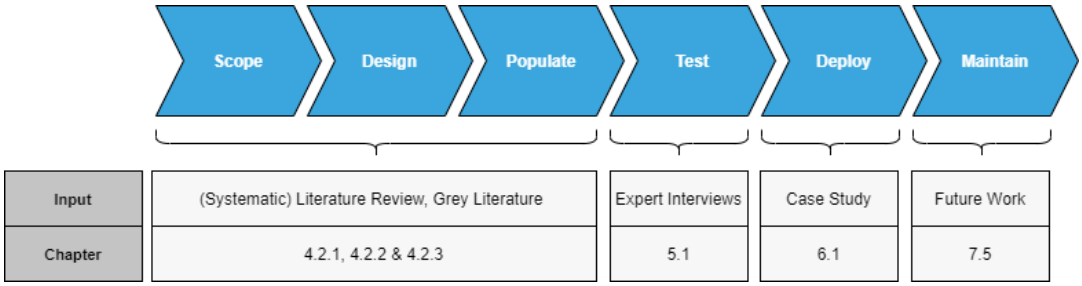



FIGURE 4.2: Development phases (adapted from De Bruin et al. (2005) for the API-m-FAMM, including input used for each phase, as well as the chapters in which they are described.

design and creation process of APIs overlaps with the topic of software engineering and would lead to a dilution of the scope of the API-m-FAMM. For example, this would result in the inclusion of practices related to development methodologies, such as Agile. However, the publish, maintain, and deprecate and retire stages of the API lifecycle are included in the scope of the API-m-FAMM. This decision is further verified during the expert interviews described in Chapter 5.1.1.

Secondly, the API-m-FAMM aims to support organizations that expose their API(s) to third-party developers in their API management activities. As a result, management of internal APIs (as per Figure 3.1) is excluded from the scope of the API-m-FAMM, in contrast to management of partner and public APIs. This is a consequence of the observation that capabilities such as developer enablement, security and analytics are of lesser importance with regards to managing internal APIs, considering that these APIs are exclusively used for internal app integration and development. However, the API-m-FAMM may still prove to be useful for such organizations wishing to incrementally improve upon capabilities such as lifecycle management, as well as increasing internal performance of their API(s). In terms of organization characteristics, we hypothesize that the API-m-FAMM will be of most value to organizations that are newly entering the API economy and wish to set up a successful API program, for example by exposing assets to third parties or by making a previously internal API publicly available and accessible. The API-m-FAMM aims to provide such organizations with a roadmap which they may use as a path for incremental improvements with regards to their API management capabilities. However, the API-m-FAMMM also seeks to assist mature organizations by providing them with more complex capabilities they might not yet have implemented.

Lastly, the API-m-FAMM seeks to provide practitioners with incremental capability improvements that are tool, technology, and platform-independent. For example, there are many tools which organizations may utilize to implement monitoring, communication, and support capabilities. In the same vein, capabilities regarding traffic management or security, such as authentication, authorization and threat protection, may be relatively straightforward to implement through the use of commercially available gateway or management platform solutions. However, some organizations may opt to develop these solutions in-house, or may not wish to use such solutions for a variety of reasons. To ensure that the API-m-FAMM may be generalized to both these types of organizations, comparisons between specific tools or management platform solutions are excluded from the scope of the model.

### 4.2.2 Design

The second phase in constructing the API-m-FAMM is concerned with the needs of the intended audience and determining how these needs will be met. De Bruin et al. (2005) states that the needs of the intended audience are reflected in why they seek to apply the model, how the model can be applied to varying organizational structures, who needs to be involved in applying the model, and what can be achieved through application of the model.

- *The 'why'* for the API-m-FAMM is that its main goal is to assist organizations that (plan to) expose their APIs to third-party developers to assess and evaluate their degree of maturity with regards to their API management capabilities.
- *The 'how'* is that organizations can utilize the API-m-FAMM to assess their as-is situation with regards to their API management capabilities, and then subsequently incrementally improve upon these capabilities by implementing practices that are of a higher maturity.
- *The 'who'* involved in applying the API-m-FAMM may vary across organizations, depending on characteristics such as their size in terms of employees involved in the API program, number of exposed APIs, and the degree of incoming traffic. For example, for a small organization that exposes one, mildly popular API, it is likely that a small number of employees are familiar with the API program and the activities involved in managing it. These employees may then utilize the API-m-FAMM to assess the as-is situation, and then use the model as a road map to incrementally implement capabilities and practices to reach a higher level of maturity. In contrast, a large organization that exposes multiple, popular APIs that generate large loads of traffic, is likely to employ multiple development teams, product owners and designers whom are involved with the API program. In this case, it is unlikely that a solitary employee or a small group of employees will be able to assess the current as-is situation of the API program. Instead, information for this assessment will have to be extracted through meetings with employees from varying teams and backgrounds whom are involved in the various aspects of API management, such as community engagement, implementing security measures, monitoring capabilities, and lifecycle management. Alternatively, consultants with a thorough understanding of API management and the API-m-FAMM may be able to apply the model by conducting interviews with all relevant stakeholders involved in the organization's API program.
- *The 'what'* that can be achieved through the application of the API-m-FAMM is an insight into the current maturity of an organization with regards to its API management capabilities, as well as a path to incremental implementation and improvement of more mature, specific practices.

### 4.2.3 Populate

Now that the scope and design of the API-m-FAMM is described, the content of the model must be decided upon. In this phase, it is necessary to identify what needs to be measured in the maturity assessment and how this can be measured (De Bruin et al., 2005). The goal is to attain the domain components, sub-components and processes the functional domain of API management consists of, which must be captured in the form of focus areas, capabilities, and practices. This process and the

resulting evolution of the API-m-FAMM as a whole and the focus areas and capabilities it is composed of is depicted in Figure 4.3. The version numbers used in this figure and the following subsections are used to describe the steps that were taken in developing the API-m-FAMM. Please note that these version numbers are not intended to imply that these versions are functional in any way, and that they are exclusively used to illustrate the evolution path of the API-m-FAMM. Furthermore, because of their abundance, only the most fundamental changes that carried over to the first version of the API-m-FAMM are discussed in the following subsections.

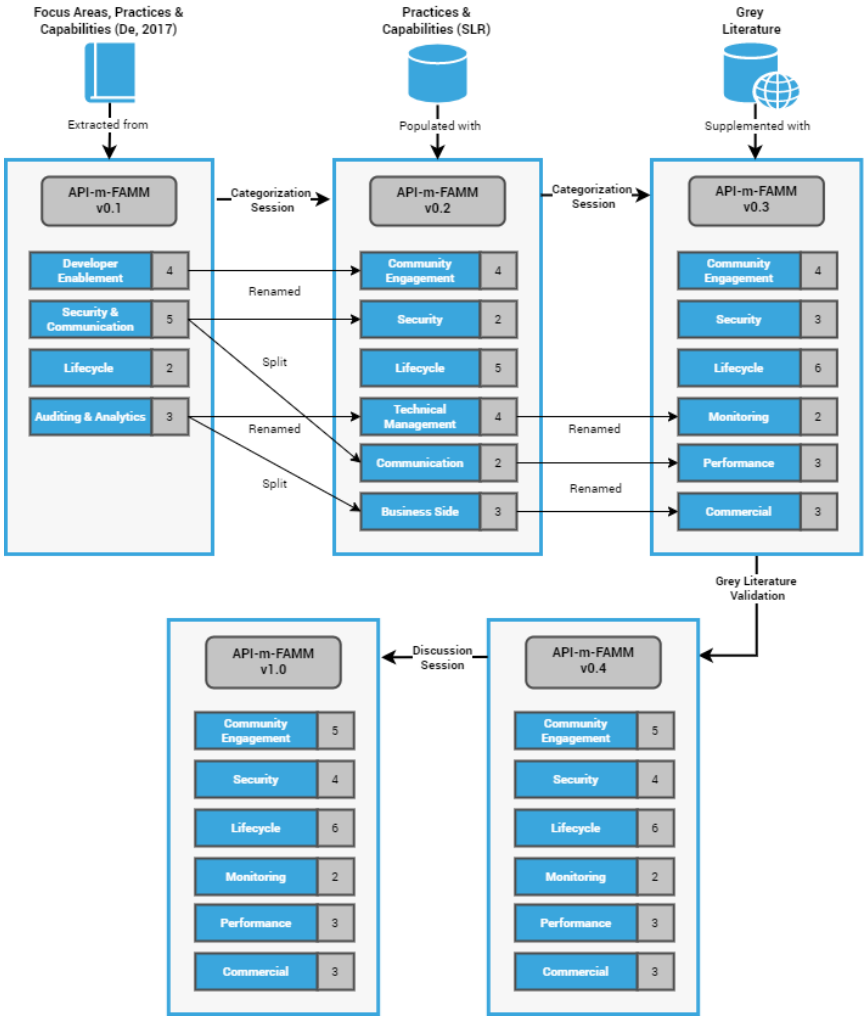



FIGURE 4.3: Evolution path of the API-m-FAMM towards its first version. The blue rectangles represent focus areas, while the grey squares represent the amount of capabilities they comprise.

### API-m-FAMM v0.1

As a first step towards populating the API-m-FAMM, a top-down approach is taken, which entails adopting the structure shown in Figure 3.4. This involves converting the lifecycle management, developer enablement, secure communications, and API auditing, logging & analytics capability categories as presented by De (2017), as well

as the respective features they comprise, into four focus areas and a set of capabilities. Individual activities identified in De's work are then converted into practices, which are then categorized and assigned to the aforementioned capabilities.

**API-m-FAMM v0.2**

This initial population and structure of the API-m-FAMM was then discussed extensively in a meeting with the author and two supervisors of this study. This session lead to the removal of a set of duplicate or redundant practices and capabilities, as well as to the conversion of some capabilities to practices, and vice-versa. Additionally, this categorization session resulted in many practices being moved to other capabilities, as well as some capabilities being relocated to other focus areas than those they were originally components of. This categorization process was conducted using *Google Drawings* as a tool. All focus areas, capabilities, and practices are represented as differently colored building blocks, accompanied by descriptions as encountered in academic literature. These were then discussed, and dragged-and-dropped to form the desired structure. Considering that the researchers were not able to meet in person due to the COVID-19 pandemic, this approach is taken as an alternative to the *Card Sorting* technique. Card sorting is a technique where individuals arrange cards representing key concepts to structure and discern relationships between concepts (Nielsen, 1995).

Now that the structure and foundation of the API-m-FAMM has been decided upon, the contents of the model are further populated with practices and capabilities. Following De Bruin et al. (2005)'s suggestion that this may best be achieved through an extensive literature review, this was accomplished by conducting a Systematic Literature Review (SLR), which is summarized in Section 1.3 and is described in detail in a separate article (Mathijssen et al., 2020a). This SLR resulted in the collection of 43 papers related to API management. Among this body of literature, 32 papers were found to contain features, activities or processes related to API Management. These were extracted in the form of focus areas, capabilities, and practices, as defined as part of the meta-model in Figure 4.1. In the scope of API management, practices were defined as *any practice that has the express goal to improve, encourage and manage the usage of APIs*. Moreover, capabilities were defined as *the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices*. The aforementioned extraction process yielded 114 practices and 39 capabilities. These practices and capabilities were used to further populate the API-m-FAMM. This was accomplished through a second categorization session. Similarly to the previous session, this resulted in the re-location of some practices and capabilities. Moreover, capabilities were added to accommodate newly added practices, such as *encryption*.

The re-location of practices and capabilities was primarily driven by the decision to split the *security & communication* focus area up into two separate focus areas: *security* and *communication*. This decision was made because security was found to be an substantial and integral topic of API management in itself. Moreover, it was decided that the communication focus area, which was later renamed to *performance*, comprises capabilities such as *service routing* that are unrelated to security. Furthermore, the decision was made to split the *auditing & analytics* focus area up into technical management, which was later renamed to *monitoring*, and business-side, which was later renamed to *commercial*. This was done due to the difference in

nature between capabilities such as *monetization* and *analytics*, which were originally grouped together. This difference was further compounded by the decision to split the traffic management capability into two separate capabilities, with one capturing the business-level aspect of this capability and the other encompassing operational aspects. The former capability was then moved to the new commercial focus area along with the monetization capability, while the latter was moved to the performance focus area.

#### API-m-FAMM v0.3

After the second categorization session, it was ultimately decided that more information was needed to determine whether practices and capabilities were suited to be included in the model with regards to their scope and relevance. In order to resolve this, the collection of practices and capabilities is verified by using information gathered from grey literature, which includes white papers, online blog posts, websites, commercial API management platform documentation and third-party tooling. Doing so resulted in the following changes made with regards to the contents of the API-m-FAMM:

- *Removal* of several practices that were found to be irrelevant, redundant, or too granular. For example, *filtering spam calls*, which was originally uncovered as part of the SLR, was found to be redundant as this practice is already covered by practices such as *DoS protection* and *rate limiting*. Consequently, such practices were removed.
- *Addition* of several practices that were newly identified. For example, *predictive analytics* was found to be a practice that is offered by multiple commercial API management platform providers. Similarly, *including change logs* was found to be a practice that is recommended by practitioners as a best practice when updating APIs. Consequently, such practices were added to the API-m-FAMM.
- *Merging* of several practices that were found to be irrelevant, redundant, or too granular. For example, practices that were originally uncovered through the SLR, such as *email-based support*, *phone-based support*, and *form-based support* were found to be redundant, as no significant difference with regards to their maturity may be discerned among these practices. Consequently, these practices were merged into one practice: *establish communication channel*.
- *Splitting* of practices that were found to be compounded by practices that were thought to warrant separate, individual practices. For example, the *black or whitelist IP addresses* was split up into the *blacklist IP addresses* and *whitelist IP addresses* practices because these were found to be relevant practices on their own. Additionally, Consequently, these practices were merged into one practice: *establish communication channel*.
- *Relocation* of practices to different capabilities than those they were originally assigned to. For example, the *Oauth2.0 authorization* practice was moved from the *authentication* capability to the newly introduced *authorization* capability as Oauth is considered to be an authorization protocol.
- *Renaming* of several practices, as well as updating descriptions and formulation of practice descriptions that were previously missing or incomplete. For

example, the *provide code samples* practice was renamed to *provide FAQ with code samples* because it was found that these two practices often go hand in hand. Additionally, this practice's description was updated.

- *Identification* of dependencies among practices, either among practices within the same capabilities or among practices across different capabilities or focus areas. Some dependencies were found to be relatively straightforward, such as the *multiple API versioning strategy* practice depending on the implementation of the *maintain multiple APIs* practice. However, dependencies between practices belonging to different capabilities such as *quota management* depending on *rate limiting* or *rate throttling* were also identified.
- *Arrangement* of practices based on their interrelated maturity with regards to the other practices in the capability they are assigned to. At this point in time, this was performed on a mostly subjective and empirical basis, and thus should be regarded as a first attempt to discern practices with regards to their relative maturity.
- *Formulation* of implementation conditions corresponding to each practice, which are aimed at providing practitioners with an overview of the necessary conditions that must be met before a practice may be marked as implemented.
- *Attachment* of the role of practitioners (as presented in Tables 3.2 and 3.3) typically involved in implementing each individual practice. However, it should be noted that these roles are merely meant to act as indicators, as naming conventions across organizations may differ and are often domain specific.

**API-m-FAMM v0.4**

After having validated the practices and capabilities contained in the API-m-FAMM through grey literature, a discussion session was conducted among the researcher and supervisors as a means to validate the contents of the model. This session resulted in some practices being rearranged with regards to their level of maturity, as well as some being removed or merged. The amount of practices and capabilities that were added, removed, merged, split, relocated or renamed as a result of the grey literature validation process and the aforementioned discussion session are shown in Table 4.2 below. However, it should be noted that some practices that were added as a result of the grey literature verification process were later removed as a result of the discussion session. As such, numbers corresponding to the *added* and *removed* operations presented in Table 4.2 are slightly inflated.

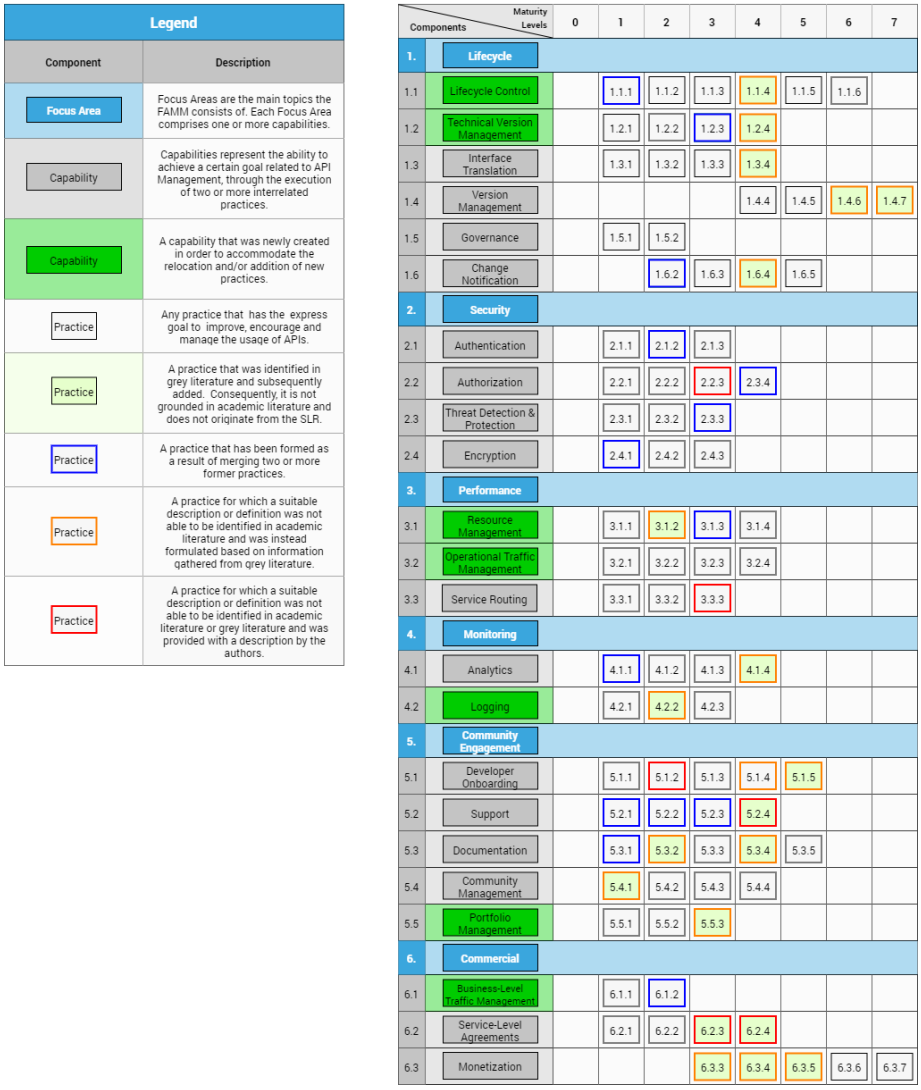
TABLE 4.2: Number of practices and capabilities added, removed, merged, split, relocated or renamed as a result of the grey literature validation process and the discussion session.

| Component | Added | Removed | Merged | Split | Relocated | Renamed |
|---|---|---|---|---|---|---|
| Practice | 17 | 27 | 39 | 4 | 12 | 93 |
| Capability | 1 | 1 | 1 | 0 | 1 | 2 |

**API-m-FAMM v1.0**

The population process outlined throughout this subsection resulted in the first version of the API-m-FAMM. In this stage of the design process, the model is grounded in academic literature, and is verified and supplemented by using information gathered from grey literature, such as white papers, online blog posts, websites, commercial API management platform documentation and third-party tooling. As a result of these activities, the initial body of 114 practices and 39 capabilities that was extracted as a result of the SLR is refined and narrowed down to 87 practices and 23 capabilities, which are divided among six focus areas. Due to their size, these focus areas, capabilities, and practices are not described in detail at this point in this work. Instead, the contents of this first version of the API-m-FAMM are elaborated upon in a separate document (Mathijssen et al., 2020b). However, the general structure of the API-m-FAMM is presented in Figure 4.4. As shown, each individual practice is assigned to a maturity level within its respective capability. Additionally, it should be noted that practices can not depend on practices as part of another capability that have a higher maturity level. For example, practice 1.4.4 is dependant on the implementation of practice 1.2.3, resulting in a higher maturity level being assigned to the former of these practices.

Figure 4.4 also shows that at this stage, 17 practices were added in addition to those extracted through the SLR. These practices were identified in the grey literature. Such practices were encountered multiple times across sources, and were deemed to be relevant for inclusion. However, it should be noted that the number of practices that were added may appear to be high when examining Figure 4.4, considering that the practices that were removed are not shown. Furthermore, 14 new practices were introduced as a result of merging 39 former practices, as shown in Table 4.2 and Figure 4.4. Additionally, 7 new capabilities were introduced as a result of the categorization and discussion sessions, in addition to those identified through the SLR. This was done to accommodate the addition and relocation of practices. Moreover, descriptions that are based on information gathered from grey literature were formulated for 18 practices for which adequate descriptions were not able to be identified in academic literature. Lastly, 6 practices are accompanied by descriptions that were formulated by the researchers themselves, as based on empirical knowledge. Even though suitable descriptions could not be identified for these practices in academic literature or through grey literature, they were included in this version of the API-m-FAMM because they were thought to be relevant for practitioners. This suspicion is tested through expert interviews (Chapter 5 and case studies (Chapter 6, which are part of the next phase in constructing the API-m-FAMM.



| Legend | |
|---|---|
| **Component** | **Description** |
| Focus Area | Focus Areas are the main topics the FAMM consists of. Each Focus Area comprises one or more capabilities. |
| Capability | Capabilities represent the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices. |
| Capability | A capability that was newly created in order to accommodate the relocation and/or addition of new practices. |
| Practice | Any practice that has the express goal to improve, encourage and manage the usage of APIs. |
| Practice | A practice that was identified in grey literature and subsequently added. Consequently, it is not grounded in academic literature and does not originate from the SLR. |
| Practice | A practice that has been formed as a result of merging two or more former practices. |
| Practice | A practice for which a suitable description or definition was not able to be identified in academic literature and was instead formulated based on information gathered from grey literature. |
| Practice | A practice for which a suitable description or definition was not able to be identified in academic literature or grey literature and was provided with a description by the authors. |

| Components \ Maturity Levels | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| 1. | Lifecycle | | | | | | | | |
| 1.1 | Lifecycle Control | | 1.1.1 | 1.1.2 | 1.1.3 | 1.1.4 | 1.1.5 | 1.1.6 | |
| 1.2 | Technical Version Management | | 1.2.1 | 1.2.2 | 1.2.3 | 1.2.4 | | | |
| 1.3 | Interface Translation | | 1.3.1 | 1.3.2 | 1.3.3 | 1.3.4 | | | |
| 1.4 | Version Management | | | | | 1.4.4 | 1.4.5 | 1.4.6 | 1.4.7 |
| 1.5 | Governance | | 1.5.1 | 1.5.2 | | | | | |
| 1.6 | Change Notification | | | 1.6.2 | 1.6.3 | 1.6.4 | 1.6.5 | | |
| 2. | Security | | | | | | | | |
| 2.1 | Authentication | | 2.1.1 | 2.1.2 | 2.1.3 | | | | |
| 2.2 | Authorization | | 2.2.1 | 2.2.2 | 2.2.3 | 2.3.4 | | | |
| 2.3 | Threat Detection & Protection | | 2.3.1 | 2.3.2 | 2.3.3 | | | | |
| 2.4 | Encryption | | 2.4.1 | 2.4.2 | 2.4.3 | | | | |
| 3. | Performance | | | | | | | | |
| 3.1 | Resource Management | | 3.1.1 | 3.1.2 | 3.1.3 | 3.1.4 | | | |
| 3.2 | Operational Traffic Management | | 3.2.1 | 3.2.2 | 3.2.3 | 3.2.4 | | | |
| 3.3 | Service Routing | | 3.3.1 | 3.3.2 | 3.3.3 | | | | |
| 4. | Monitoring | | | | | | | | |
| 4.1 | Analytics | | 4.1.1 | 4.1.2 | 4.1.3 | 4.1.4 | | | |
| 4.2 | Logging | | 4.2.1 | 4.2.2 | 4.2.3 | | | | |
| 5. | Community Engagement | | | | | | | | |
| 5.1 | Developer Onboarding | | 5.1.1 | 5.1.2 | 5.1.3 | 5.1.4 | 5.1.5 | | |
| 5.2 | Support | | 5.2.1 | 5.2.2 | 5.2.3 | 5.2.4 | | | |
| 5.3 | Documentation | | 5.3.1 | 5.3.2 | 5.3.3 | 5.3.4 | 5.3.5 | | |
| 5.4 | Community Management | | 5.4.1 | 5.4.2 | 5.4.3 | 5.4.4 | | | |
| 5.5 | Portfolio Management | | 5.5.1 | 5.5.2 | 5.5.3 | | | | |
| 6. | Commercial | | | | | | | | |
| 6.1 | Business-Level Traffic Management | | 6.1.1 | 6.1.2 | | | | | |
| 6.2 | Service-Level Agreements | | 6.2.1 | 6.2.2 | 6.2.3 | 6.2.4 | | | |
| 6.3 | Monetization | | | | 6.3.3 | 6.3.4 | 6.3.5 | 6.3.6 | 6.3.7 |

FIGURE 4.4: The first version of the API-m-FAMM and the focus areas, capabilities, and practices it consists of. Additionally, it is shown which capabilities and practices were newly introduced between API-m-FAMM v0.2 and v1.0, as well as for which practices descriptions were formulated based on grey literature. Please consult the legend on the top left-hand side of the figure for more information regarding the differently shaped and/or colored components.

# Chapter 5

# Evaluating the API-m-FAMM

Now that the API-m-FAMM has been populated, it is ready to be evaluated. In order to do so, De Bruin et al. (2005) state that it is important to test both the construct of the model, as well as the model instruments for validity, reliability and generalisability. Construct validity is represented by face and content validity, the former of which is measured by assessing to what degree translation of the constructs have been achieved. De Bruin et al. (2005) argue that this may best be accomplished during population of the model, through tools such as focus groups and interviews. In the case of the API-m-FAMM, this is done in the form of categorization and discussion sessions in which all researchers were involved, as well as additional bi-weekly meetings with one of this work's supervisors to ensure inter-rater agreement. According to De Bruin et al. (2005), face validity may be achieved through the selection of complementary population methods. This is accomplished through the means of grey literature, as is described in the previous subsection and shown in Figure 4.3. Content validity is measured by assessing the completeness of the domain representation. De Bruin et al. (2005) suggest that the extent of the literature review conducted and breadth of the domain covered provide a solid measure with regards to content validity. For the API-m-FAMM, an extensive systematic literature review and grey literature is used to ensure content validity.

## 5.1 Test

In order to further evaluate the API-m-FAMM's construct and content validity, two evaluation cycles are conducted, as is depicted in Figure 5.1. This evaluation is done through expert interviews, with experts being selected based on the criteria outlined in Chapter 2.2.4. The first evaluation cycle, which is described in Chapter 5.1.1, consists of 11 semi-structured interviews that are conducted with 9 different experts, with the main goal of collecting suggestions for addition, removal, or relocation of practices and capabilities. Furthermore, the first version of the API-m-FAMM is evaluated in terms of its *completeness*, *operational feasibility*, *ease of use*, *usefulness*, and *effectiveness* during this first cycle. The second evaluation cycle, which is described in Chapter 5.1.2, consists of 3 unstructured interviews with experts that are deemed to have the broadest knowledge base with regards to API management. These experts are selected from the sample size of 9 experts that were involved in the first evaluation cycle. This second evaluation cycle is conducted in order to evaluate whether the way in which suggestions for changes that were made by experts during the first cycle were interpreted correctly.

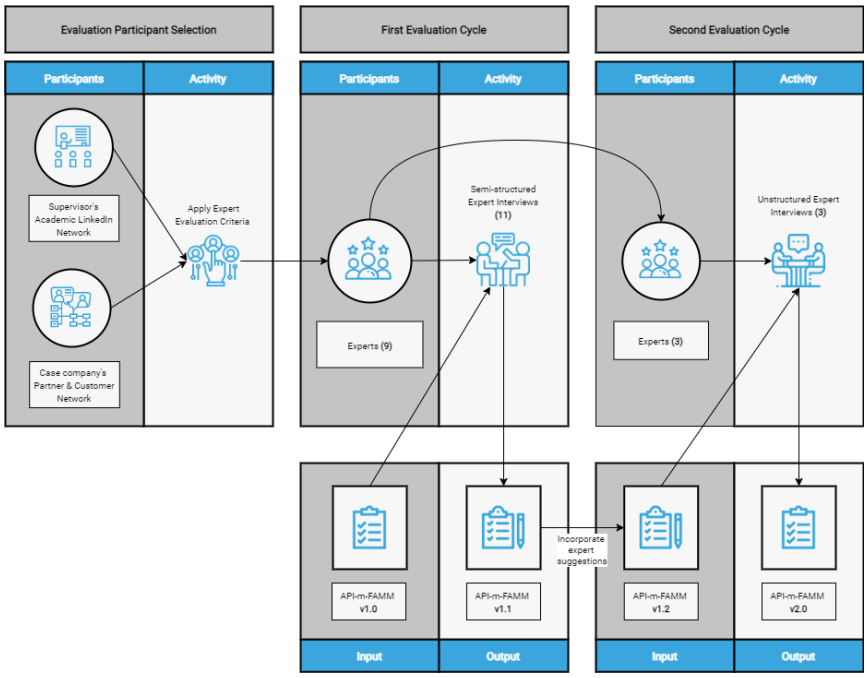



FIGURE 5.1: Overview of the expert selection process, as well as the cycles that are conducted in evaluating the API-m-FAMM. Additionally, it is shown which versions of the API-m-FAMM is used as input for interviews, as well as the versions that are produced as a result.

### 5.1.1 First Evaluation Cycle

In order to further evaluate the API-m-FAMM's construct and content validity, a series of expert interviews are conducted. As is described in Chapter 2.2.4, these interviews are aimed at evaluating the first version of the API-m-FAMM based on a set of evaluation criteria, the first of which being *completeness*. During interviews, this criterion is measured by a set of open questions. These questions are displayed in the 'content evaluation phase' section of the interview protocol, which may be reviewed in Appendix B. In addition to being measured to a more thorough extent as part of the case study described in Chapter 6, the *operational feasibility*, *ease of use*, *usefulness*, and *effectiveness* criteria are also evaluated during interviews. These criteria are measured with a set of likert scale questions, which are listed in the 'closing phase' section of the aforementioned interview protocol. Answers given in response to these questions are discussed in Chapter 5.1.1 and shown in Table 5.3.

As can be seen in Table 5.1, interviews are conducted with 9 experts. Summaries of these interviews as well as the experts they were conducted with may be found in Appendix C. These experts are selected as based on the selection criteria outlined in Chapter 2.2.4. Based on an interviewee's knowledge regarding the six focus areas the API-m-FAMM consists of, which is measured through the survey shown in Appendix B, one or two focus areas are selected for evaluation. Focus areas are evaluated through 11 interviews, which result in each focus area being evaluated three times. During these interviews, which are semi-structured in nature, the API-m-FAMM in its entirety is first presented to the expert. Next, the focus area that is selected for evaluation and each capability it comprises is described. Then, all practices these capabilities consist of are elaborated upon. For each capability, experts

are asked whether they are familiar with it and whether they believe it is assigned to the correct focus area. Similarly, experts are asked whether they are familiar with each practice, and whether they believe it is assigned to the correct capability. Additionally, they are asked whether they can identify any dependencies with regards to the implementation of other practices. After having answered these questions for a capability and the practices it comprises, experts are asked to rank practices in terms of their perceived maturity and complexity for each capability. This ranking exercise is performed in Google Drawing by using the same card sorting technique that was used to initially structure the API-m-FAMM, as explained in Section 4.2.3. In order to illustrate this ranking exercise, the maturity levels of practices belonging to the *Logging* capability as perceived by three experts is shown in Figure 5.2.

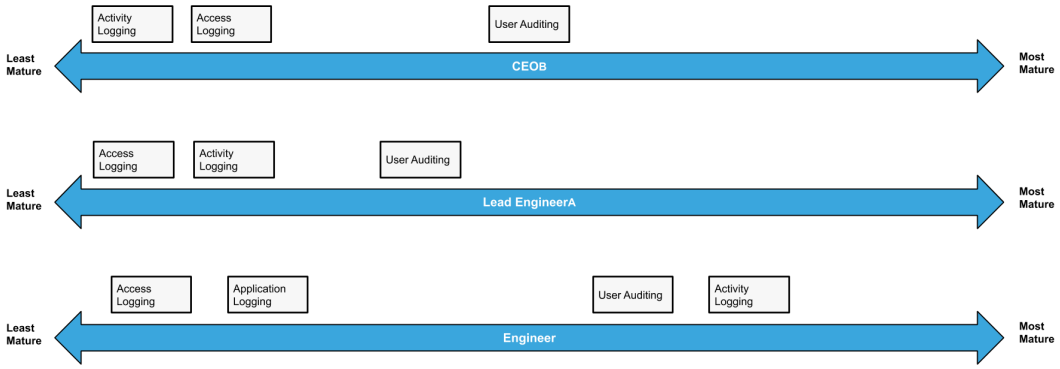



FIGURE 5.2: Assigned maturity levels corresponding to the practices belonging to the logging capability, as perceived by three experts, placed along an ordinal scale ranging from least mature to most mature.

In the event where an expert suggested the addition of a new practice, this practice was added. The expert was then asked to rank this new practice in terms of its perceived maturity alongside the already existing practices. For example, it can be seen in Figure 5.2 that the *Application Logging* practice was suggested to be added by *Engineer* during the third interview that was focused on evaluating the *Monitoring* focus area. Furthermore, the newly suggested practice was discussed in subsequent interviews that were focused on the same focus area the practice was suggested to be assigned to. Similarly, upon suggesting the removal of a practice, this practice was then excluded from the ranking exercise and the suggestion for removal was discussed in subsequent interviews.

### Interview Results (API-m-FAMM v1.1)

As was mentioned earlier, 11 expert interviews were conducted. During these interviews, many additions and changes in terms of the API-m-FAMM's structure and contents were suggested by experts, whom were encouraged to elaborate on their motivation regarding these suggestions. As a result of transcribing and processing the recordings of all interviews, numerous suggestions that were made by experts to either add, remove, merge, split, relocate, or rename several focus areas, capabilities, and practices, are compiled. The amount in which these suggestions for changes occurred are shown in Table 5.2 below, as grouped by the type of suggested

TABLE 5.1: Interviewees and the current role they fulfill within their organization, as well as the years of experience they have with either consuming, developing, integrating, providing,versioning, monitoring or managing APIs. Additionally, it is shown which focus areas were discussed with interviewees, as well as the duration of the interviews.

| Interviewee | Experience | Hours | Community | Security | Lifecycle | Monitoring | Performance | Commercial |
|---|---|---|---|---|---|---|---|---|
| *API Evangelist* | 10+ | 1 | ✓ | | | | | ✓ |
| $CEO_A$ | 10+ | 1.5 | ✓ | | | | | |
| $CEO_B$ | 10+ | 5.5 | | ✓ | ✓ | ✓ | | |
| *Engineer* | 10+ | 1 | | | | ✓ | ✓ | |
| *IT Consultant* | 10+ | 3.5 | ✓ | ✓ | ✓ | | | ✓ |
| *Product Manager* | 6 | 3 | | ✓ | | | | |
| *Lead* $Engineer_A$ | 10+ | 1 | | | | ✓ | ✓ | |
| *Lead* $Engineer_B$ | 10+ | 1 | | | | | ✓ | ✓ |
| *Lead* $Engineer_C$ | 5 | 1.5 | | | ✓ | | | |
| **Total** | N/A | 19 | 3 | 3 | 3 | 3 | 3 | 3 |

change as well as the type of component they apply to. Additionally, these changes are visually represented in their entirety in Figure 5.3, along with the number of experts that suggested for a specific change to be made. However, it should be noted that this is an intermediary version of the API-m-FAMM that is made with the sole purpose of representing all suggested changes in a visual manner, so that it may be used as input for future discussion sessions. These sessions are described in the next subsection, and are aimed at interpreting and processing the suggestions. Furthermore, API-m-FAMM v1.1 comprises all focus areas, capabilities, and practices that are contained in API-m-FAMM v1.0. These components are marked with either an uninterrupted border that is colored black, orange, or red, or a dashed border that is colored blue or purple. Additionally, API-m-FAMM v1.1 also incorporates all suggestions that were made for the addition for new practices, capabilities, and focus areas. These are marked with an uninterrupted green border.

Evidently, the number of practices that were suggested to be added is high. However, it should be noted that while a large part of these practices were explicitly mentioned by experts, some were also indirectly extracted from transcripts as a result of comments the expert had made. Additionally, no suggestions are rejected at this point, hence all suggestions that were made by experts are taken into account and incorporated into Table 5.2 and Figure 5.3. The process of extracting suggestions from the interview transcripts was purposely done in an inclusive manner. This decision is made to ensure that all the different viewpoints and opinions voiced by experts are considered as part of future discussions. Furthermore, the researchers are of the opinion that the high amount of suggestions for addition of practices highlights the added value and importance of conducting expert interviews in the scope of this research. Considering that the version of the API-m-FAMM that was used as input for the interviews is exclusively based on findings from academic literature and grey literature, the fact that a large amount of practices was uncovered through expert interviews aids in adequately supplementing the model with industry-based knowledge. This is necessary to ensure that this research's objective is achieved, which is to construct an API management maturity assessment framework that is grounded in both literature and industry. Furthermore, the quality of the API-m-FAMM is not negatively affected by including all suggestions made by experts at this point in the development process. All suggestions are extensively analyzed and discussed (as is described in the next subsection) by the researchers, as well as being evaluated during the second evaluation cycle and case studies, before being included in the final

version of the model.

TABLE 5.2: Number of practices, capabilities, and focus areas that were suggested to be added, removed, merged, split, relocated or renamed by experts during interviews.

| Component | Added | Removed | Merged | Split | Relocated | Renamed |
|---|---|---|---|---|---|---|
| **Practice** | 50 | 5 | 3 | 3 | 9 | 3 |
| **Capability** | 7 | 0 | 0 | 2 | 2 | 2 |
| **Focus Area** | 1 | 0 | 0 | 0 | 0 | 3 |

### Changes and Decisions Made (API-m-FAMM v1.2)

After having compiled all suggestions made by experts, six extensive discussion sessions are held among all researchers to analyze, discuss, and interpret them. During each session, one focus area is selected to be discussed. All suggested changes to either the focus area itself, or the capabilities or practices it consists of are then analyzed and interpreted through the help of the transcribed arguments that were provided by experts during the interviews. As a result, numerous modifications are made to the API-m-FAMM, which are visualized in its entirety in Figure 5.4. Additionally, some fundamental decisions are made with regards to the scope and contents of the API-m-FAMM.

- Firstly, it was decided that all practices that are contained in the model should be implementable *without* the usage of an API management platform. This decision was made due to several reasons. First of all, it was found that among the organizations that the experts that were consulted are employed at, only a small portion actively utilizes a third party platform to manage their API(s). When asked, experts belonging to the category that have not incorporated an API management platform into their organizations cited arguments such as wanting to avoid vendor lock-in, high costs, or simply not having a need for many of the functionalities provided by such management platforms. Oftentimes, the latter argument was tied to the organization currently exclusively using internal APIs, thus removing the need for using a management platform to manage and expose any partner or public APIs altogether. Considering that it may reasonably be hypothesized that these arguments may likely also apply to other organizations wishing to consult the API-m-FAMM to evaluate and improve upon their API management related practices, any practices or capabilities that were found to be directly tied to the usage of an API management platform were removed from the model. For example, this was the case for the *Visual Data Mapping* practice, which is exclusively provided by the *Axway* API management platform [1], as well as the practices corresponding to the newly suggested *Error Handling* capability, which are implementable through the use of the *Apigee* platform [2].

  An additional reason for excluding such capabilities and practices is that they are likely to evolve throughout the coming years, which would in turn require the API-m-FAMM to be updated as well. In order to prevent this, the API-m-FAMM and the practices it comprises should be platform-independent. Lastly,

[1] https://www.axway.com/en/products/api-management
[2] https://cloud.google.com/apigee/api-management?hl=nl

the purpose of the API-m-FAMM is not to guide practitioners in selecting an appropriate commercial API management platform for their organization. Instead, the API-m-FAMM aims to guide organizations in assessing and evaluating their current maturity in terms of those processes that are considered to be best-practices and are at the core of API management, so that they may then develop a strategy towards implementing practices that are currently not implemented and desirable in further maturing the organization in terms of API management.

- Secondly, many practices were deemed to be too granular, specific, or irrelevant to be included. Consequently, such practices were either removed, or merged into a practice that is composed of these smaller practices. An example of practices that were found to be too granular include newly suggested practices such as *Event Participation*, *Event Hosting*, and *Organize Hackathons*. Additionally, since determining a difference among these practices in terms of their maturity was found to be unfeasible, they were instead merged into the *Organize Events* practice and included in its description.

- Thirdly, some practices that describe a specific protocol were renamed to be more ambiguous and generic. For example, the former *OAuth 2.0 Authorization* practice was renamed to *Standardized Authorization Protocol*, with a referral to the OAuth 2.0 protocol being included in its description instead. This was done to ensure that the API-m-FAMM remains functional and applicable in the future, since it is likely that new protocols will be developed and adopted among the industry in the future. These concerns also applied to suggested practices corresponding to individual authentication methods such as client certificate and SAML authentication, which were ultimately merged into the *Implement Authentication Protocol* practice and included in its description. An additional reason for doing so in the case of these authentication methods is that they each have their individual strengths and weaknesses, with one not always necessarily being 'better' or more mature than the other. Furthermore, some methods may be more appropriate for some use cases than others.

- Furthermore, some capabilities and its corresponding practices that were also thought to apply to most organizations in general, and that are not necessarily involved with API management, were excluded from the model. An example of this is the *Financial Management* capability that was suggested to be added. Considering that practices such as *Automated Billing*, *Third-Party Payment Provider Integration*, and *Revenue Sharing* are best practices that apply to commercially oriented organizations in general, they were removed. This decision was made to ensure that the contents of the API-m-FAMM is exclusively composed of practices that are directly tied to API management.

- During interviews focused on the *Lifecycle* focus area, experts were asked to elaborate on the manner in which their organization has implemented *Governance* (as is described in Chapter 3.7). Based on the answers given however, it became clear that capturing processes related to governance in the form of practices is not feasible. This may largely be attributed to the observation that such processes seem to be inherent to specific characteristics of the organization, such as its culture, size, usage of a third party API management platform, as well as the amount of APIs that are used or exposed by the organization.

Some practices were suggested for addition by *IT Consultant*, such as *Define Naming Conventions*, *Define Best Practices*, and *Define Integration Patterns*. However, after having discussed these with experts in subsequent interviews, it was decided that these practices are too abstract and inconcrete in comparison with other practices, considering that they may be interpreted in different ways by practitioners due to the varying organizational characteristics mentioned earlier. Hence, the *Governance* capability that was originally part of the *Lifecycle* focus area was removed, along with the *Design-time Governance* and *Run-time Governance* practices it was composed of.

- A valuable suggestion that was made by experts is the addition of monitoring in terms of the amount of resources that calls to the API consume, such as CPU, disk, memory, and network usage. Considering that this monitoring perspective was previously missing alongside performance and health monitoring, as well as it being suggested by multiple experts independently from one another, the *Resource Monitoring* practice was newly added. Similarly, this resource perspective was also found to be missing among the *Traffic Management* capability, alongside the *Request Limiting* and *Request Throttling* practices. Hence, the *Data Volume Limiting* practice was newly added.

- Another fundamental change that was made to the API-m-FAMM is the renaming of the former *Monitoring* focus area to *Observability*. This rename was independently suggested by two experts, whom argued that observability better describes the focus area, considering that the *Analytics* capability was split into two capabilities: *Monitoring* and *Analytics*. This decision was made because experts were of the opinion that monitoring is concerned with gathering (real-time) metrics related to the API's health, performance, and resource usage, while analytics is concerned with aggregating these metrics so that insights may be formed and subsequent action may be taken based off of these. As a result, the monitoring capability was added, as well as practices related either to monitoring or analytics being moved to the capabilities they are associated with.

- Moreover, some practices that were originally posed from a passive perspective, were changed with the intention of being conducted in an active manner. For example, the *Include Changelogs* practice was renamed to *Distribute Changelogs*, and its description was changed so that its focus is changed from passive inclusion of changelogs in the reference documentation, to active distribution of changelogs to consumers of the API. Similarly, the *Provide API Status Page* was renamed to *Broadcast API Status*, as well as its description being changed to signify the operational status of the API being broadcasted to consumers in an active manner, as opposed to providing an API status page in a passive fashion. These changes were made due to the fact that when phrased in a passive manner, these practices were deemed to be too irrelevant to be included in the API-m-FAMM, considering that the level of maturity required to implement these practices is too low when compared to other practices. When phrased from an active perspective however, these practices can be considered to be best practices that an organization should strive to implement.

- Finally, a major fundamental change was made with regards to the *Lifecycle Control* capability. While practices belonging to this capability, such as *API Endpoint Creation*, *API Publication*, and *Import Pre-existing API*, are considered

to be an integral aspect of API management in both literature as well as the industry, the decision was made to exclude these practices from the API-m-FAMM. This choice was made due to the fact that being able to design, create, publish, and deploy an API is a precondition for implementing all other practices the model consists of. Moreover, during interviews it became clear that it was difficult for experts to rank these practices in terms of their maturity, considering that they are often performed in chronological order.

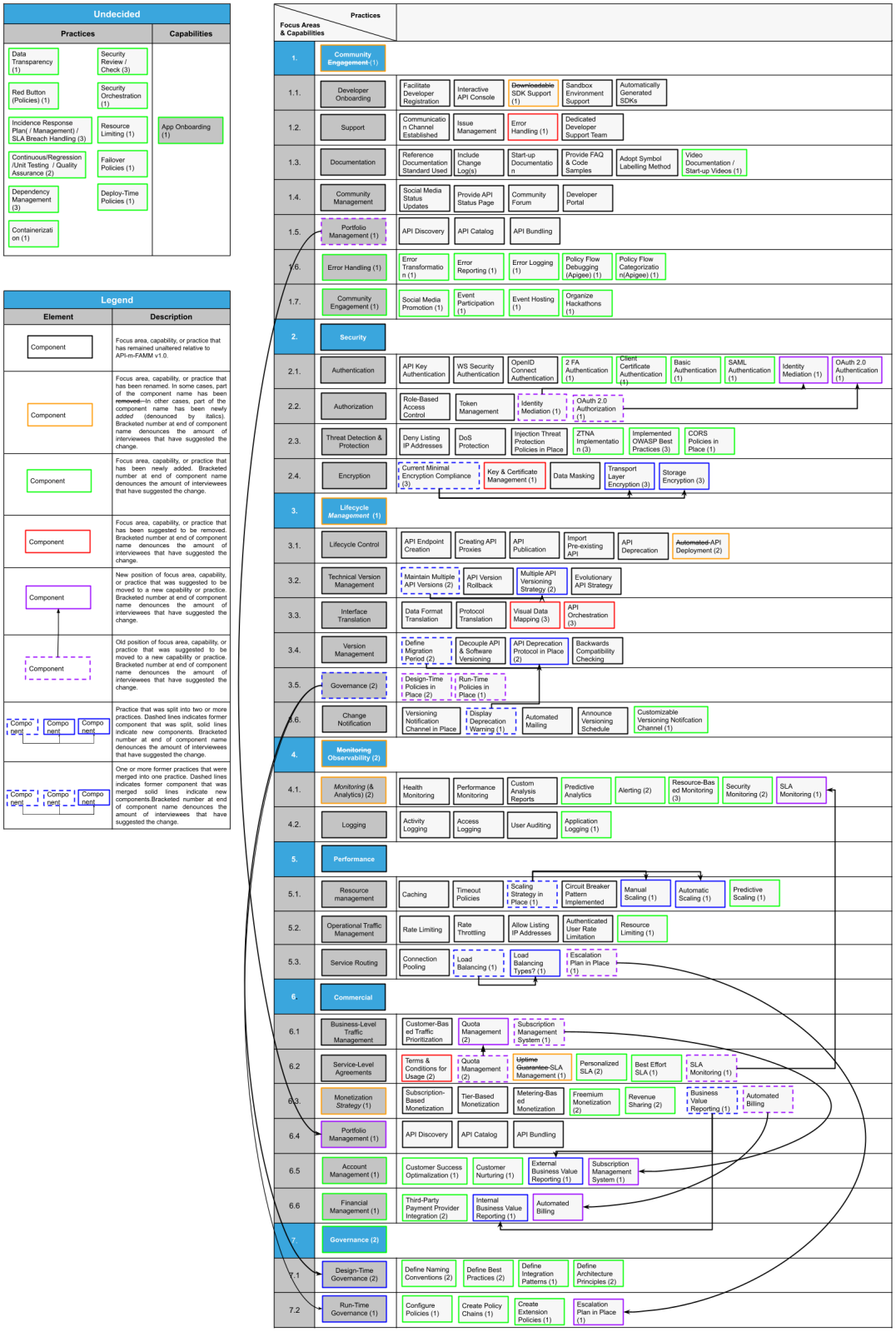



FIGURE 5.3: API-m-FAMM v1.1, which includes all suggested changes that were made by experts during interviews. Please consult the legend on the left-hand side of the figure for more information regarding the manner in which the colored outlines should be interpreted. Practices and capabilities that were not directly categorized by the expert during interviews are placed in the 'undecided' box on the top-left hand side.

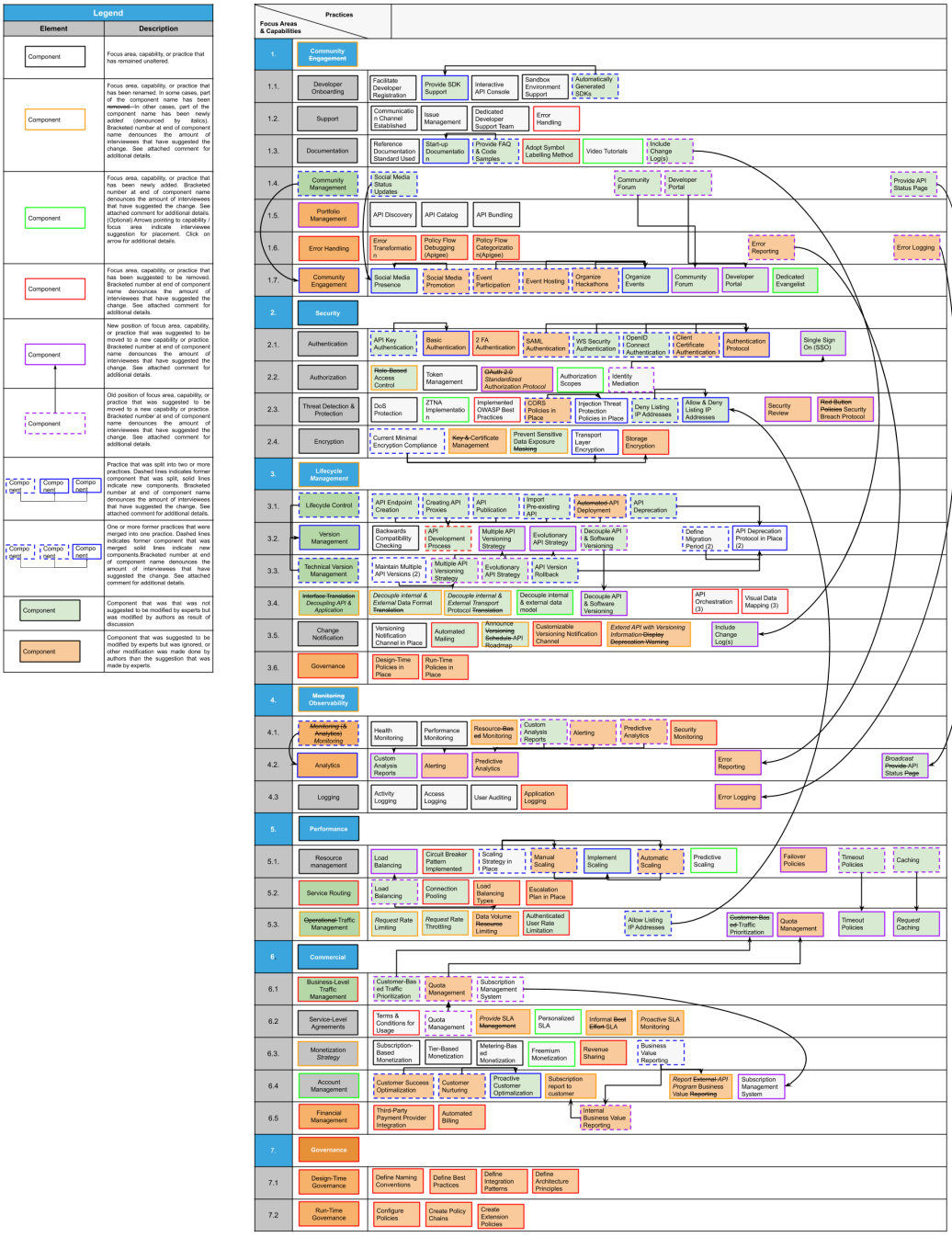



FIGURE 5.4: API-m-FAMM v1.2, including all suggested changes that were made by experts during interviews, as well as the manner in which they were subsequently interpreted and applied by the researchers. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which the colored outlines and fills should be interpreted.

### Evaluation Criteria Results

In addition to its contents and *completeness*, the first version of the API-m-FAMM is also evaluated with regards to its *operational feasibility*, *ease of use*, *usefulness*, and *effectiveness*, as is shown in Table 2.3. These criteria are measured by asking experts to rank the API-m-FAMM on a likert-scale ranging from 1 to 5 in response to the following questions, which are posed at the end of the interview:

- **Operational Feasibility**: How likely do you think it would be that an organization would actually use the API-m-FAMM in practice to evaluate and improve upon their API management related processes?
- **Ease of Use**: How easy do you think it would be to understand the API-m-FAMM's content and use it to self-assess and evaluate your organization's maturity in API management?
- **Usefulness**: How useful do you think the API-m-FAMM would be in providing you and your organization with valuable and interesting insights in your organization's API management related processes?
- **Effectiveness**: How effective do you think the API-m-FAMM would be in helping you and your organization improve on their API management related processes?

The way in which the API-m-FAMM was ranked by each expert in response to these questions is summarized in Table 5.3 below. However, these results should be prefaced by a few initial remarks. Firstly, it should be noted that during the interviews, an intermediate and unfinished version of the API-m-FAMM was presented to experts. One of the implications of this is that maturity levels were absent from this version of the model. Because of this, experts often found it difficult to envision what the final version would look like, as well as to properly judge its ultimate capabilities and potential for helping an organization in improving their API management maturity. Additionally, due to time constraints, only one or two of the six focus areas the model consists of were selected for discussion, which, in some instances, impaired experts' ability to grasp the API-m-FAMM's scope as a whole on a conceptual level. Furthermore, the descriptions given for the focus areas, capabilities, and practices during the interview were summarized and shortened. Lastly, the experts had not familiarized themselves with the contents of the API-m-FAMM prior to the interview taking place.

TABLE 5.3: The rankings given by all experts in response to the questions corresponding to the four evaluation criteria, as well as their averages.

| Interviewee | Operational Feasibility | Ease of Use | Usefulness | Effectiveness |
|---|---|---|---|---|
| *API Evangelist* | 3 | 2 | 4 | 3 |
| $CEO_A$ | 4 | 3 | 4 | 4 |
| $CEO_B$ | 3 | 2 | 3 | 4 |
| *Engineer* | 4 | 4 | 4 | 5 |
| *IT Consultant* | 2 | 2 | 3 | 4 |
| *Product Manager* | 4 | 4 | 3 | 3 |
| *Lead* $Engineer_A$ | 4 | 3 | 5 | 4 |
| *Lead* $Engineer_B$ | 3 | 2 | 4 | 3 |
| *Lead* $Engineer_C$ | 4 | 3 | 5 | 4 |
| **Average** | 3.4 | 2.8 | 3.9 | 3.8 |

In addition to the remarks mentioned above, experts cited other reasons that help in explaining the relatively low scores that were given for the operational feasibility and ease of use criteria in particular. For example, the following quote is extracted from the interview that was conducted with *IT Consultant*, whom ranked the API-m-FAMM's operational feasibility and ease of use with a 2:

> *"I think there are some pretty complex terms being used in this model. I think the ease in which organizations are able to self-assess their processes depends on how well all factors are described and whether people understand. I give workshops and trainings which include explanations, but still people often have trouble understanding the terminology used."*

Possibly, this point of criticism could be partially alleviated by ensuring that practitioners wishing to employ the API-m-FAMM have fully read through all descriptions of practices and capabilities, which are more elaborate than those shown during interviews. This argument is further supported by the following quotes by $CEO_A$ and $Lead\ Engineer_B$:

> *"It took me a while to understand the contents of the model, but eventually I did."*

> *"I find it hard to judge the model's ease of use because we are skimming through at a rapid pace. Also, I think it might be hard for organizations to assess themselves, because a lot of explanation is needed for them to understand what is meant with each process. However, I can imagine a good consultant can work with this very well. The included practices are not incomprehensible, but people who are not familiar with the subject will probably need some time to fully read through everything and such."*

A possible way in which these concerns could be resolved is, as $Lead\ Engineer_B$ also mentions in the quote shown above, to employ a consultant whom is well-versed in the domain of API management and is fully familiar with the API-m-FAMM and its contents. However, we hypothesize that the self-assessment will be made significantly easier if multiple practitioners within the organization, whom each possess knowledge on the six focus areas the API-m-FAMM consists of, participate in the assessment.

With regards to the API-m-FAMM's perceived usefulness and effectiveness, the majority of experts ranked the model relatively high. In particular, some experts were of the opinion that the API-m-FAMM could be helpful for smaller organizations and organizations that are (thinking of) starting to expose an API. For example, *Engineer* and $Lead\ Engineer_A$ argued:

> *"I think the model is very useful. At my organization, we are so large with so much traffic that we have figured out and implemented most of these practices, but I think there are many organizations that have no idea what a good API is. For example with regards to rate limiting, I think you would be amazed to see that many APIs would go down if they receive too much traffic, purely because it not has been implemented."*

> *"I think it is useful, because I think there are a lot of teams that are struggling with these things. This model can help them with checking whether they have forgotten anything. Especially when you are about to expose an API publicly I think it is ideal to have such a model."*

However, *IT Consultant* is of the opinion that the true value of the API-m-FAMM lies in its potential to be used for offering advice on any practices that the organization under evaluation has yet to implement:

> *"I think it depends on the advice that follows from the evaluation. If you do a maturity study and assess the current maturity of the organization, it would be nice if it can also be explained what processes are missing, what benefits may be achieved if those processes are implemented, and how implementation may be achieved. However, I do think this is a very helpful first step towards achieving this. I would be very interested in using this checklist for my own work.*

**Maturity Level Assignment (API-m-FAMM v2.0)**

Now that the API-m-FAMM has been evaluated and all suggested changes have been analyzed and interpreted, practices are able to be assigned to individual maturity levels. This is done by using the results of the maturity ranking exercises that were described in Section 5.1 as input. First however, all dependencies between practices are identified, which are depicted in Figure 5.5. In this context, a dependency entails that one or more practices that the practice in question is dependant on are required to be implemented before the practice may be implemented. These dependencies may either occur; (1) between practices within the same capability; (2) between practices that are assigned to different capabilities within the same focus area, or (3) between practices that are assigned to different capabilities and focus areas. In total 34 dependencies are identified, which was done by analyzing academic literature stemming from the SLR, grey literature, and input received through expert interviews and the discussion sessions that were conducted among the researchers. The number of dependencies that are identified are shown for each focus area in Table 5.4, as well as for each of the three dependency types mentioned.

TABLE 5.4

| Focus Area | Within Capability | Within Focus Area | Between Focus Areas | Total |
|---|---|---|---|---|
| Community | 3 | 0 | 0 | 3 |
| Security | 2 | 0 | 0 | 2 |
| Lifecycle Management | 3 | 1 | 2 | 6 |
| Observability | 0 | 6 | 0 | 6 |
| Performance | 4 | 0 | 2 | 6 |
| Commercial | 2 | 1 | 8 | 11 |
| **Total** | 14 | 8 | 12 | 34 |

As an example of a dependency between practices within the same capability, implementation of the *Implement Load Balancing* practice is required before the *Implement Scaling* practice may be implemented. An example of a dependency between practices that are assigned to different capabilities within the same focus area is the dependency between *Enable Predictive Analytics* and *Performance Monitoring*. The former practice belongs to the *Analytics* capability, while the latter practice belongs to the *Monitoring* capability, but both capabilities are contained within the *Observability* focus area. An example of a dependency between practices that are assigned to different capabilities and focus areas may be observed in the case of the dependency between the *Adopt Metering-based Monetization Model* and *Resource Monitoring* practices. The former practice is assigned to the *Monetization Strategies* capability within the *Commercial* focus area, while the latter practice is assigned to the *Monitoring* capability within the *Performance* focus area.

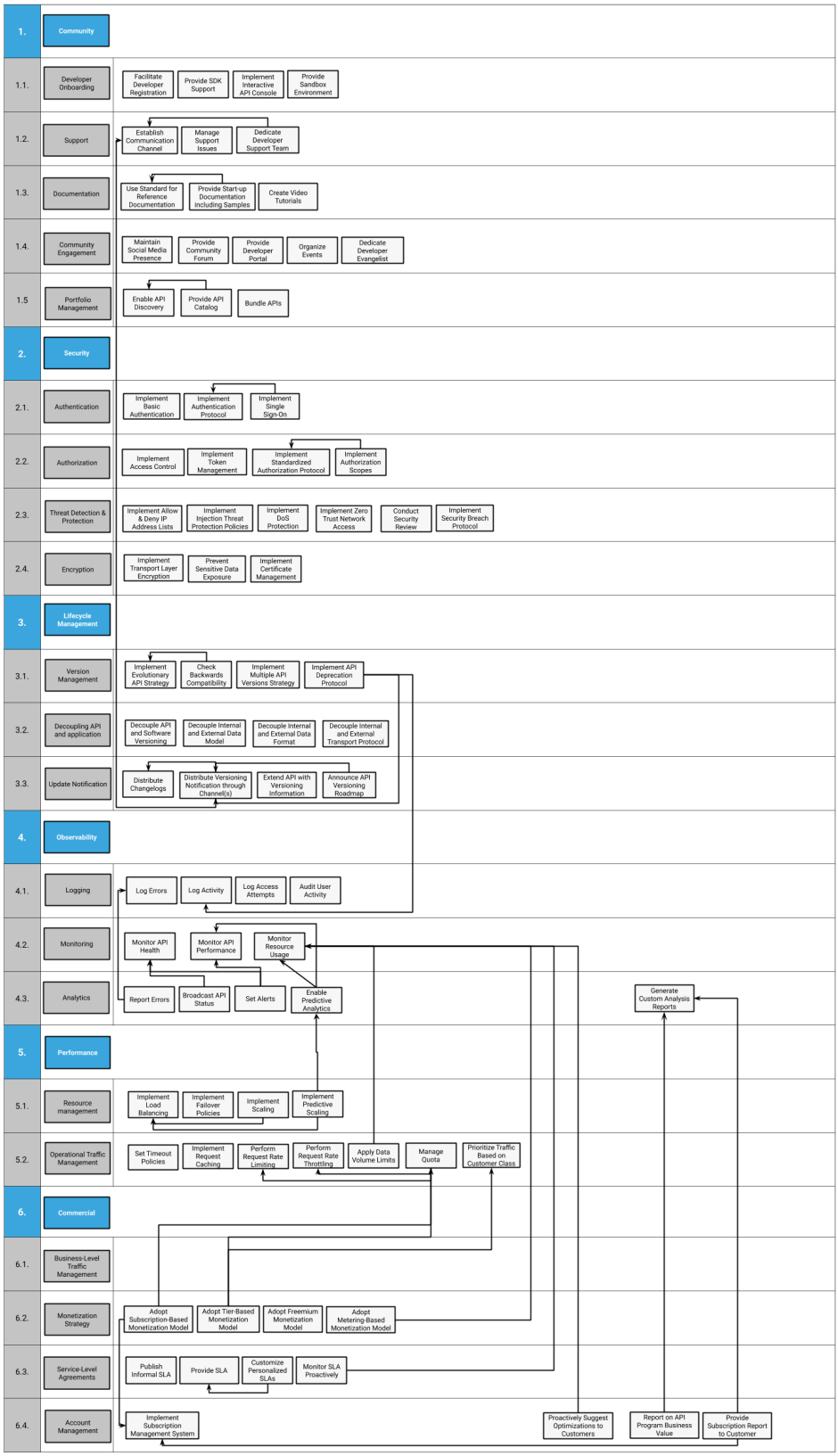



FIGURE 5.5: The API-m-FAMM after all changes had been applied, showing all dependencies that were identified between practices. In order to improve legibility, practices are not ranked in terms of their maturity in this figure.

After having identified all dependencies between practices, all 34 practices that have one or more dependencies are juxtaposed in a matrix. This is done by adhering to the constraint that practices can not depend on practices that have a higher maturity level. As a result, the foundation of the API-m-FAMM is formed, with practices ranging from maturity levels 1 to 10. Using this structure as a base, all other practices are subsequently assigned to individual maturity levels within their respective capabilities. These assignments are performed by using the results of the maturity ranking exercises that were performed by experts as one of the main sources of input, as is described in Section 5.1.

By again using the *Logging* capability as an example, the interpretation of such a maturity ranking exercise is visualized in Figure 5.6. In this figure, it can be seen that the *Activity Logging*, *Access Logging*, and *User Auditing* practices were ranked by 3 experts in terms of their perceived maturity. An additional practice, *Application Logging*, was suggested for addition by *Engineer*. However, this practice was removed because the decision was made to exclude applications in terms of abstraction from the API-m-FAMM, which is why it is outlined in red. Additionally, the decision was made to include and move the *Error Logging* practice to the *Logging* capability. Hence, this practice is outlined in green, and is included in this ranking exercise by incorporating this practice in the figure, along with the capability it was originally categorized with by the expert. Furthermore, the *Error Reporting* practice was moved to the *Analytics* capability (as can be seen in Figure 5.4, which is why it is outlined in purple and excluded from this maturity ranking exercise. Lastly, the remaining 3 practices that were suggested to be added are excluded, along with the *Error Handling* capability as a whole, which is denoted by the red outlines.

Furthermore, arrows are included that range from the lowest a practice has been ranked in terms of its perceived maturity, to its highest. Dotted lines are attached to each practice, which are then connected to these arrows with a small circle in order to highlight and compare the maturity assignments of each expert with one another. Subsequently, dashed lines are used to indicate a rough estimate of the average of these assignments, which are then mapped on the maturity levels. However, it should be noted that Figure 5.6 was made for illustratory purposes, in order to provide the reader with a conceptual idea of the manner in which the maturity assignments were performed. In practice, the maturity assignment of practices was done in a pragmatic manner, through discussion sessions among the researchers during which the expert's varying maturity rankings and their accompanying motivation and arguments were discussed and interpreted. Based on the outcome of these discussions, decisions were then made to assign practices to individual maturity levels, while taking the experts' opinions and maturity rankings into account.

Next, all practices are renamed to fit an uniform syntactical structure, which starts with a verb, followed by one or more nouns. For example, *User Auditing* is renamed to *Audit Users*, and *Resource Monitoring* is renamed to *Monitor Resource Usage*. Furthermore, descriptions of the practices that are included in the API-m-FAMM after all changes had been applied are updated. When possible, this is done using information and input that was provided by experts during interviews. Ultimately, these activities produced a second, updated version of the API-m-FAMM, which is shown in Figure 5.7 and consists of 6 focus areas, 20 capabilities, and 81 practices. These are described in Appendix D as well as in a separate source document (Mathijssen, Overeem, & Jansen, 2021a).

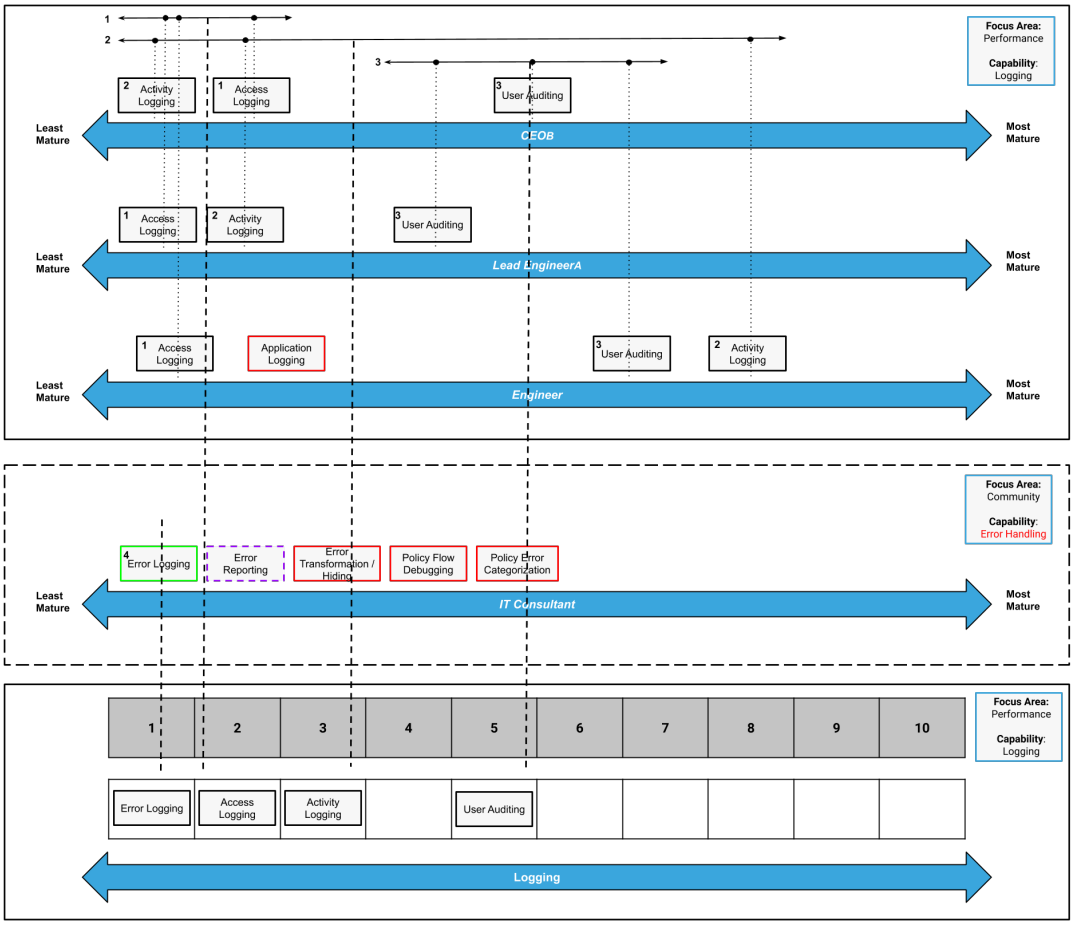
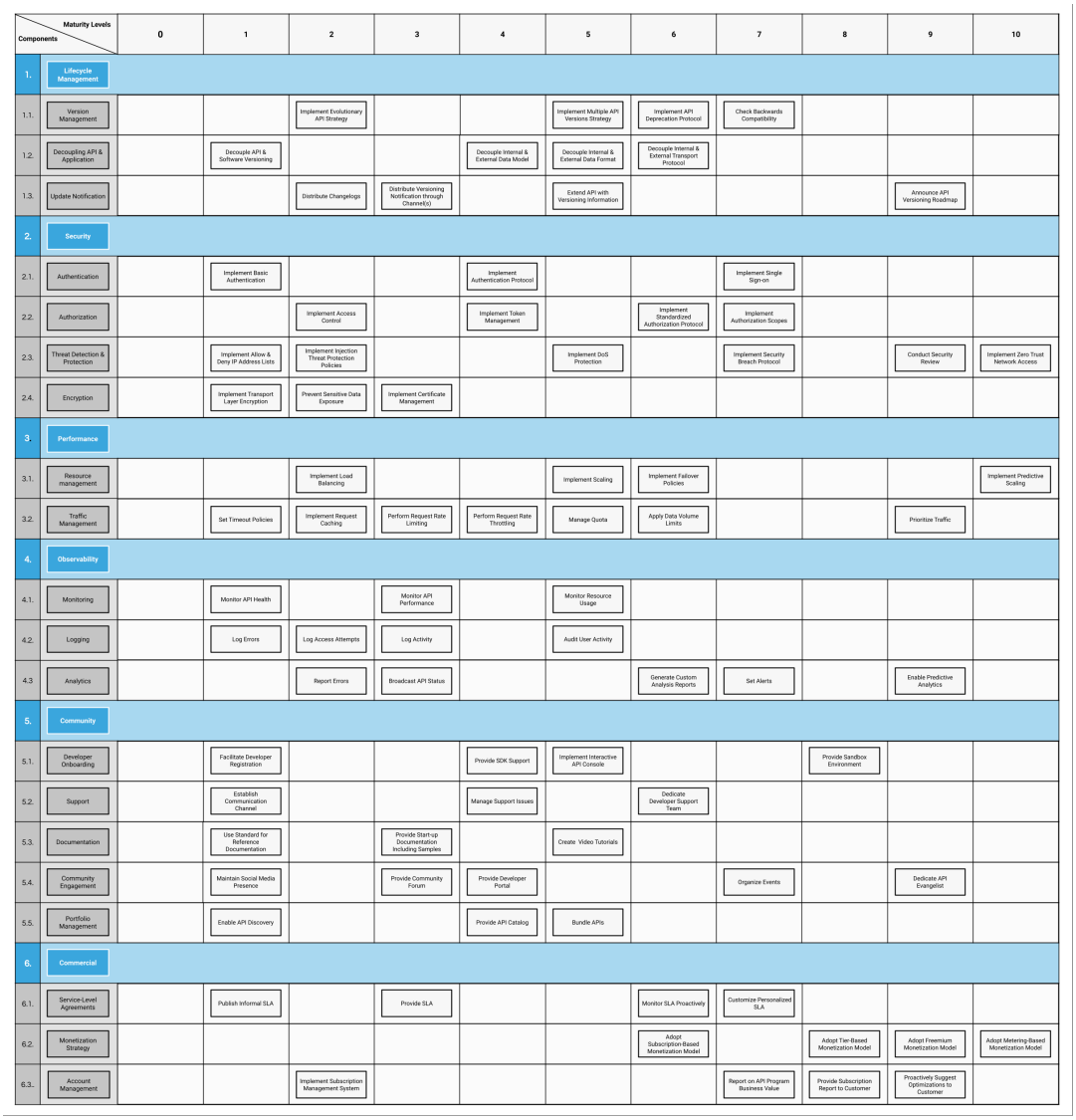



FIGURE 5.6: Conceptual overview representing a rough approximation of the way in which the expert's maturity rankings were interpreted and used as a starting point for performing the maturity level assignments.

### 5.1.2 Second Evaluation Cycle

After having updated the API-m-FAMM to incorporate all findings from the previously conducted interviews, a second evaluation cycle is conducted. This is done as a means for evaluating and verifying whether experts agree with the fundamental decisions that were made (as discussed in Section 5.1.1), as well as gathering feedback on the way suggestions made by experts were interpreted and the maturity levels that practices have been assigned to. This second evaluation cycle consists of unstructured interviews with three experts originating from the same sample of experts that were interviewed during the first evaluation cycle: *Product Manager*, *IT Consultant*, and *Lead Engineer*$_A$. During these interviews, the changes made as a result of the previous evaluation cycle, as well as the newly introduced maturity assignments are presented and discussed.

Since all experts agreed with the fundamental decisions that were made, no major further adjustments are made to the API-m-FAMM as a result of this evaluation cycle. In general, the three experts responded positively to the model, and expressed interest in using it in practice to evaluate their organization's API management related processes and assess their API management maturity. For example, *Product*



| Components \ Maturity Levels | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1. Lifecycle Management** | | | | | | | | | | | |
| 1.1. Version Management | | | Implement Evolutionary API Strategy | | | Implement Multiple API Versions Strategy | Implement API Deprecation Protocol | Check Backwards Compatibility | | | |
| 1.2. Decoupling API & Application | | Decouple API & Software Versioning | | | Decouple Internal & External Data Model | Decouple Internal & External Data Format | Decouple Internal & External Transport Protocol | | | | |
| 1.3. Update Notification | | | Distribute Changelogs | Distribute Versioning Notification through Channel(s) | | Extend API with Versioning Information | | | | Announce API Versioning Roadmap | |
| **2. Security** | | | | | | | | | | | |
| 2.1. Authentication | | Implement Basic Authentication | | | Implement Authentication Protocol | | | Implement Single Sign-on | | | |
| 2.2. Authorization | | | Implement Access Control | | Implement Token Management | | Implement Standardized Authorization Protocol | Implement Authorization Scopes | | | |
| 2.3. Threat Detection & Protection | | Implement Allow & Deny IP Address Lists | Implement Injection Threat Protection Policies | | | Implement DoS Protection | | Implement Security Breach Protocol | | Conduct Security Review | Implement Zero Trust Network Access |
| 2.4. Encryption | | Implement Transport Layer Encryption | Prevent Sensitive Data Exposure | Implement Certificate Management | | | | | | | |
| **3. Performance** | | | | | | | | | | | |
| 3.1. Resource management | | | Implement Load Balancing | | | Implement Scaling | Implement Failover Policies | | | | Implement Predictive Scaling |
| 3.2. Traffic Management | | Set Timeout Policies | Implement Request Caching | Perform Request Rate Limiting | Perform Request Rate Throttling | Manage Quota | Apply Data Volume Limits | | | Prioritize Traffic | |
| **4. Observability** | | | | | | | | | | | |
| 4.1. Monitoring | | Monitor API Health | | Monitor API Performance | | Monitor Resource Usage | | | | | |
| 4.2. Logging | | Log Errors | Log Access Attempts | Log Activity | | Audit User Activity | | | | | |
| 4.3. Analytics | | | Report Errors | Broadcast API Status | | | Generate Custom Analysis Reports | Set Alerts | | Enable Predictive Analytics | |
| **5. Community** | | | | | | | | | | | |
| 5.1. Developer Onboarding | | Facilitate Developer Registration | | | Provide SDK Support | Implement Interactive API Console | | | Provide Sandbox Environment | | |
| 5.2. Support | | Establish Communication Channel | | | Manage Support Issues | | Dedicate Developer Support Team | | | | |
| 5.3. Documentation | | Use Standard for Reference Documentation | | Provide Start-up Documentation Including Samples | | Create Video Tutorials | | | | | |
| 5.4. Community Engagement | | Maintain Social Media Presence | | Provide Community Forum | Provide Developer Portal | | | Organize Events | | Dedicate API Evangelist | |
| 5.5. Portfolio Management | | Enable API Discovery | | | Provide API Catalog | Bundle APIs | | | | | |
| **6. Commercial** | | | | | | | | | | | |
| 6.1. Service-Level Agreements | | Publish Informal SLA | | Provide SLA | | | Monitor SLA Proactively | Customize Personalized SLA | | | |
| 6.2. Monetization Strategy | | | | | | | Adopt Subscription-Based Monetization Model | | Adopt Tier-Based Monetization Model | Adopt Freemium Monetization Model | Adopt Metering-Based Monetization Model |
| 6.3. Account Management | | | Implement Subscription Management System | | | | | Report on API Program Business Value | Provide Subscription Report to Customer | Proactively Suggest Optimizations to Customer | |

FIGURE 5.7: API-m-FAMM v2.0, which includes the assignment of all practices to their respective maturity levels, which range from level 1 to level 10.

*Manager* concluded the interview with the following remark:

> *"I think this - the API-m-FAMM - is a very thorough analysis. You have made a very nice overview that can help organizations with deciding what and when they have to do when wanting to start with an API. If they want to bring something to the market quickly, this helps them realize they must first have implemented the processes on the lower levels, and have to start small. Actually, for many organizations that already have an API or want to start building one, this is the roadmap they should follow for a good API strategy."*

# Chapter 6

# Case Study

As is described in Section 2.2.4, a case study design is employed in order to answer SRQ4. This case study is evaluative in nature and is aimed at determining to what degree the API-m-FAMM succeeds in aiding an organization in evaluating and improving upon their API management related business processes in practice. This case study consists of two parts: an embedded, single-case study and a multiple case study. These classification types adhere to those presented by Yin et al. (2003). The embedded single-case study is visualized in Figure 6.1, while the multiple case study is depicted in Figure 6.2.

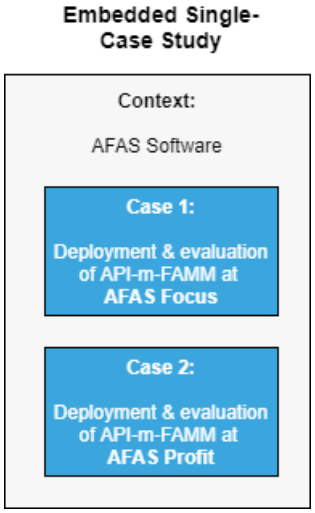



FIGURE 6.1: Schematic overview of the embedded single-case study design.

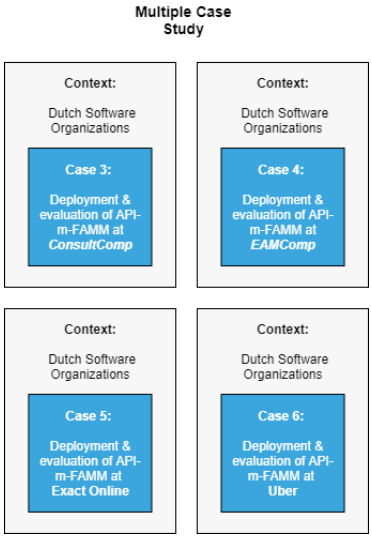



FIGURE 6.2: Schematic overview of the multiple case study design.

For the embedded single-case study, the API-m-FAMM is applied to two software products that are developed by AFAS software. These products are described in further detail in Section 6.1.1. The multiple case study is conducted at four case companies, which are described in Section 6.1.2. Data resulting from the application of the API-m-FAMM is collected through an Excel spreadsheet. Finally, participants evaluated the API-m-FAMM on the same criteria that were employed as part of the first evaluation cycle. These processes are part of the case study protocol that is used in conducting the case studies, which is described in Section 6.1.3. The results that are extracted from the deployment of the API-m-FAMM as part of these case studies are reported on in Chapter 6.2. Next, the results of the case study as a whole are reflected upon, discussed, and compared to one another in Chapter 6.3. Finally, in

Chapter 6.4, changes that are made as a result of the case study findings are elaborated on and the resulting final version of the API-m-FAMM is presented.

## 6.1 Deploy

The aforementioned embedded, single-case study is conducted at AFAS Software, which is described in the following subsection. Within this case company, the API-m-FAMM was deployed and evaluated with two development teams that are working on two separate software products. The multiple case study is conducted with multiple organizations. The largest part of these organizations are selected by contacting the experts that were previously interviewed as part of the first evaluation cycle. Furthermore, other organizations are selected through the utilization of the academic contact network of the supervisors of this work. Both parts of the case study are described in further detail below, by first introducing the case company and the two development teams and software products the API-m-FAMM is evaluated with. Next, the various organizations that are involved in the multiple case study are described.

### 6.1.1 Embedded Single-Case Study Company

AFAS is a Dutch vendor of ERP software based in Leusden, the Netherlands. Additionally, AFAS has offices in Belgium and Curaçao. The privately held company currently employs over 500 people and annually generates €191 million of revenue.

#### AFAS Profit

AFAS' main software product is Profit, which is an ERP package consisting of different modules such as Fiscal, Financial, HRM, Logistics, Payroll, and CRM. Currently, this product has over 2 million users across over 11.000 small, medium and large organizations. The latest version currently on the market is called Profit 17, which was released in 2020 and includes both a windows application as well as a web based version. The ERP system is offered as Software-as-a-Service (SaaS), called AFAS Online, or can be hosted by the client on premise.

AFAS Profit provides customers with two APIs: a REST API and a SOAP API. Both of these APIs offer the same functionalities, and customers may decide on using either of these depending on their preferences. Functionalities are offered through two endpoint types, which AFAS calls *Connectors*: Get connectors and Update connectors. The former type of endpoints enable external applications to retrieve data from the AFAS database, while the latter is used to enable external applications to add, change, or remove data from the database. Combined, these endpoints are called about 500 million times a month. Furthermore, standard connections with external software products and applications that utilize these connectors and that AFAS is partnered with are offered through AFAS' partner portal.

#### AFAS Focus

Currently, AFAS is developing a new version of its ERP software, which is called Focus. This product is being developed from scratch in terms of the technology and codebase used, and is cloud-based as well as generated by using the ontological model of an enterprise as input. Furthermore, it will form the main foundation for generating an entire software suite on a cloud infrastructure platform of choice,

considering that Focus is entirely platform- and database-independent. Additionally, Focus is envisioned to enable rapid model-driven application development and will drastically increase customization flexibility for AFAS' partners and customers. Considering that since the time development first commenced some modules such as Financial have been developed, AFAS is currently in the process of transferring customers from the current Profit product to Focus.

Conceptually, Focus offers the same two endpoints as Profit: Get Connectors and Update Connectors. However, these are only available as REST APIs, which support both the XML and JSON data formats. The endpoints are described using the OpenAPI specification and make use of the OAuth application token flow for authentication. The current size and state of Focus encompasses about twenty available endpoints, which are directed at a few specific integrations. In the future this number will increase as Focus continues to grow and branch out to more partners.

### 6.1.2 Multiple Single-Case Study Companies

In this section, the organizations that have participated in the multiple single-case study are briefly described. This is done by summarizing key characteristics such as the types of services provided, as well as the amounts of customers that are served and the amount of employees that work for the organizations. However, please note that the names of some of these organizations are anonymized. Explicit consent to include the name of the organization was obtained of those that are not.

#### ConsultComp

*ConsultComp* is a multinational professional services network, is one of the Big Four accounting organisations, and is among the largest professional services networks in the world by revenue and number of professionals. *ConsultComp* provides audit, consulting, financial advisory, risk advisory, tax, and legal services with over 300.000 professionals globally. Aside from these services, the organization also develops software products for customers, which are developed in-house in their office in the Netherlands. The team involved in developing these products utilizes internal APIs, third-party service integrations to access data from service providers, and also is in the process of starting to expose (partner) APIs to customers.

#### EAMComp

*EAMComp* is an organization that is based in the Netherlands and that provides customers with an Enterprise Asset Management (EAM) platform. This platform is cloud-based and comprises features such as maintenance, safety, medical asset, infra Asset, IT Service, and facility management, enabling customers to manage their organization's physical assets. Currently, over 100.000 customers utilize *EAMComp*'s platform. The platform provides customers with a REST API, with which they are able to retrieve data from *EAMComp*'s database and integrate with partner solutions.

#### Exact

Exact is a multinational organization that provides ERP software and was founded in the Netherlands. Apart from their headquarters in the Dutch city Delft, Exact also has offices in 20 other countries, and currently employs over 1850 people and

annually generates €209 million of revenue. Exact provides customers with various products, such as an integrated ERP package, and a package that incorporates modules that are targeted towards CRM, HRM, and workflow management.

Exact's main software product that is offered in the Netherlands and Belgium is called Exact Online, which is a package consisting of modules such as accountancy, CRM, and project management. This SaaS product currently has over 500.000 users and is fully internet-based. Exact Online provides customers with two main API types: a REST API and a XML API. These APIs comprise a range of endpoints which combined are called about 700 million times a month.

#### Uber

Uber is a large-scale multinational technology organization that was founded in the United States. It provides multiple services, such as ride-hailing, food delivery (Uber Eats), and package delivery. Uber is estimated to have over 93 million monthly active users worldwide. In 2012, Uber established its international headquarters in Amsterdam, the Netherlands. Here, among others, development teams are responsible for optimizing the performance and scalability of the public API that is provided by the organization, as well as developing new functionalities.

### 6.1.3 Case Study Protocol

As mentioned earlier, the main objective of the case study is to determine to what degree the API-m-FAMM succeeds in aiding an organization in evaluating and improving upon their API management related business processes in practice. In order to do so, data regarding the organization's API management processes needs to be collected. This is done by providing the selected organizations with a visual copy of the API-m-FAMM (as depicted in Figure 5.7), as well as a copy of the source document (Mathijssen et al., 2021a) describing the focus areas, capabilities, and practices the model consists of in detail. However, considering that the API-m-FAMM was previously presented to selected participants as part of the first and/or second evaluation cycle (with the exception of *EAMComp*), they were already informed with regards to the focus of this research as well as the purpose and structure of the API-m-FAMM.

After having familiarized themselves with the contents of the API-m-FAMM, participants are asked to use the model to first determine what focus areas and respective capabilities are relevant within their organization. Next, by first starting at maturity level 1 and then moving on towards maturity level 10, participants are asked to assign each practice to one of five possible implementation categories. These categories are inspired by the practice implementation categories that are used as part of case studies conducted by Jansen (2020) in evaluating his Focus Area Maturity Model for software ecosystem governance. In this work, four categories are introduced: *implementable*, *implementable and planned*, *not applicable*, and *not implementable*.

For this research's case study, one additional category is added: *implemented*. This category is introduced to enable data collection regarding the as-is situation of practice implementation in the participating case companies. Having insight into the current status of implementation is paramount towards ensuring that the API-m-FAMM succeeds in achieving its goal, which is to assess the organization's current level of maturity. Additionally, the *implementable* category that is used in Jansen

(2020)'s work is renamed to *implementable but not planned*. This is done to highlight and clarify the difference between this category and the *implementable and planned* category. The resulting five practice implementation categories that the practitioners are able to choose from are defined as follows:

- **Implemented**: the practice has currently been implemented in the organization.
- **Implementable but not planned**: the practice has not been implemented, but in theory is implementable. However, the practice is not currently planned for implementation in the foreseeable future.
- **Implementable and planned**: the practice has not been implemented, but has been planned for implementation in the foreseeable future.
- **Not implementable**: the practice has not been implemented, and is not able to be implemented within the organization.
- **Not applicable**: the practice has not been implemented, and is not applicable; for example, the practice may not be of added value or desirable for the organization to implement.

Collection of this data is enabled through the use of an Excel spreadsheet, as shown in Appendix E, which participants are provided with and asked to return by email once it had been filled out.

Lastly, participants are asked to fill out a short survey, which is presented in Appendix E and consists of two parts. First, participants are asked to specify to which organization or software product the answers they have provided through the spreadsheet apply. Additionally, they are asked whether they are comfortable with the publication of the name of their organization in this work, and whether they would like to take part in a follow-up interview to discuss their answers given in the previously filled in spreadsheet.

The second part of the survey consists of a series of questions that are similar to those that were asked as part of the expert interviews during the first evaluation cycle. The purpose of these questions is for the participants of the case study to evaluate the API-m-FAMM with regards to its *operational feasibility*, *ease of use*, *usefulness*, and *effectiveness*. However, while the questions that were asked at the end of the expert interviews during the first evaluation cycle were formulated in the future tense, the questions posed as part of this evaluation survey were formulated in the past tense. These questions, which are presented below, are aimed at determining the degree to which the the API-m-FAMM has succeeded in aiding the participating organization in evaluating and improving upon their API management related business processes in practice.

- **Operational Feasibility**: Are you expecting that you or your organization will use the API-m-FAMM in practice again to evaluate and improve on your API management related processes, so that you can track your progress?
- **Ease of Use**: How easy did you find it to use the API-m-FAMM to self-assess and evaluate your organization's maturity in API management?
- **Usefulness**: How useful do you think the API-m-FAMM was in providing you and your organization with valuable and interesting insights in your organization's API management related processes?

- **Effectiveness**: How effective do you think the API-m-FAMM is in helping you and your organization improve on their API management related processes?

## 6.2 Results

In this chapter, the results of the maturity assessment of API management practices at the various case companies using the API-m-FAMM are discussed. For each case company, a visual representation of the implementation status that was assigned to each practice by the practitioner is shown. However, it should first be noted that as a result of analyzing the results of the case studies, changes were made to some of the five practice implementation categories that were introduced in the previous section. Specifically, the *not implementable* and *not applicable* categories were merged into the *not applicable* category. Additionally, the *implementable but not planned* and *implementable and planned* categories were merged into a implementation category named *implementable*. Practices that practitioners had originally marked as *not implementable* in the data collection spreadsheet were changed to *not applicable* by the researchers, and practices marked as *implementable and planned* or *implementable but not planned* were changed to *implementable*. These changes were made after all data had been collected, and are elaborated upon in further detail in Chapter 6.4. Furthermore, all changes that were made to the practice implementation categories are summarized and visualized in Figure 6.3. Moreover, considering that the descriptions of the new practice implementation categories were updated to reflect these changes and are used in discussing the results of the case studies throughout the remainder of this chapter and Chapter 6.3, the new descriptions are shown below.

- **Implemented**: the practice has currently been implemented in the organization.
- **Implementable**: the practice has not been implemented, but in theory is implementable. Depending on the organization's needs and plans, the practice will either be implemented in the short-term, long-term, or not at all.
- **Not applicable**: the practice has not been implemented, and is not applicable as it will most likely never be implemented. This may be caused by a number of reasons. For example, the practice may not be of added value, or not desirable for the organization to implement because it does not align with the organization's goals, vision, or needs.

The remainder of this chapter is structured as follows. First, results from the embedded single-case study regarding AFAS' Profit and Focus products are presented. Next, results from the multiple single-case study companies are discussed, with results of a case company of which its practitioner misinterpreted the case study protocol being discussed separately.

### 6.2.1 Embedded Single-Case Study Company Results

As part of the single-case study, AFAS' two software products Profit and Focus, as described in the previous section, are assessed on their maturity with regards to API management. As a first step, the scope and goal of the API-m-FAMM as well as the steps involved in the case study protocol were presented to the involved practitioners. After these steps had been executed and the filled-in spreadsheet had been

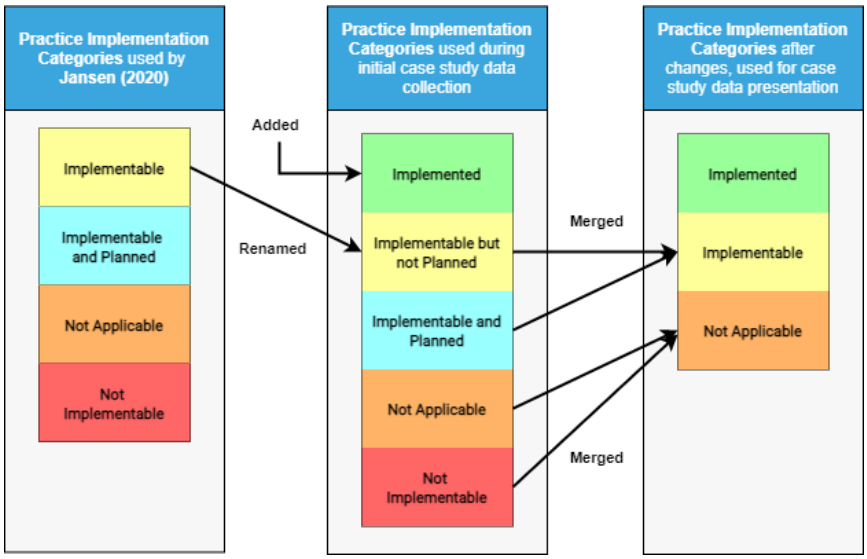


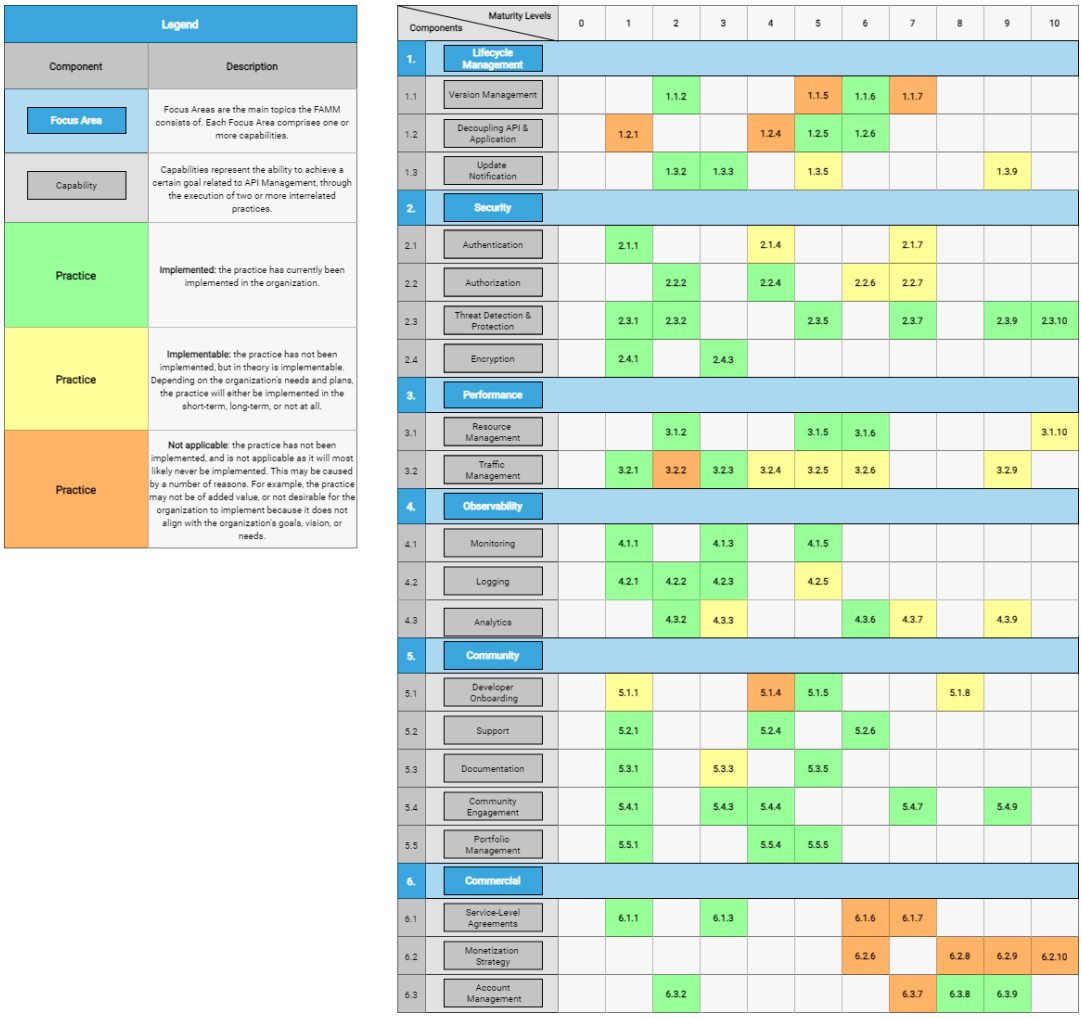

FIGURE 6.3: The manner in which the 4 practice implementation categories that were originally introduced by Jansen (2020) were adapted to fit this work's case study. The categories were first extended to 5 categories, which were used during the initial case study data collection, and subsequently merged into 3 categories as a result of the discussions that were conducted during the embedded single-case study.

returned to the researchers, a semi-structured followup interview between the researchers and the practitioners was conducted to gather feedback and to inquire about any perceived inconsistencies that were encountered among the provided answers. The results of the assessments are elaborated on for both of the two individual products in the following subsections.

### AFAS Profit

For AFAS' Profit product, the API-m-FAMM was applied by two practitioners: the manager of the Test & Quality team, and the manager of the ICT department. The statuses corresponding to the implementation of each practice as extracted from the spreadsheet provided by practitioners are visualized in Figure 6.4.

With regards to the *Lifecycle Management* focus area, several practices were marked as *not applicable*. In particular, this is the case for the *Implement Multiple API Versioning Strategy* practice. When asked, the practitioners noted that this strategy has not yet been necessary in versioning the APIs. This is due to the fact that the version of the connectors are directly tied to the version of the database on the back-end. However, events in the past have occurred where functional modifications or additions had been made to the update connectors, resulting in consumers having to change their implementation. In these events, these consumers had to be notified. Furthermore, in order to notify consumers of updates, Profit publishes release notes describing general updates made to the product as well as specific updates made to the connectors. Additionally, such changes are communicated proactively to partners through mailing. As a next step, AFAS is considering using RSS, an ECS feed, or warning headers to inform consumers ahead of time when maintenance or updates will take place.



| Legend | |
|---|---|
| Component | Description |
| Focus Area | Focus Areas are the main topics the FAMM consists of. Each Focus Area comprises one or more capabilities. |
| Capability | Capabilities represent the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices. |
| Practice | **Implemented:** the practice has currently been implemented in the organization. |
| Practice | **Implementable:** the practice has not been implemented, but in theory is implementable. Depending on the organization's needs and plans, the practice will either be implemented in the short-term, long-term, or not at all. |
| Practice | **Not applicable:** the practice has not been implemented, and is not applicable as it will most likely never be implemented. This may be caused by a number of reasons. For example, the practice may not be of added value, or not desirable for the organization to implement because it does not align with the organization's goals, vision, or needs. |

| Components \ Maturity Levels | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Lifecycle Management | | | | | | | | | | | |
| 1.1 | Version Management | | | 1.1.2 | | | 1.1.5 | 1.1.6 | 1.1.7 | | | |
| 1.2 | Decoupling API & Application | | 1.2.1 | | | 1.2.4 | 1.2.5 | 1.2.6 | | | | |
| 1.3 | Update Notification | | | 1.3.2 | 1.3.3 | | 1.3.5 | | | | 1.3.9 | |
| 2. | Security | | | | | | | | | | | |
| 2.1 | Authentication | | 2.1.1 | | | 2.1.4 | | | 2.1.7 | | | |
| 2.2 | Authorization | | | 2.2.2 | | 2.2.4 | | 2.2.6 | 2.2.7 | | | |
| 2.3 | Threat Detection & Protection | | 2.3.1 | 2.3.2 | | | 2.3.5 | | 2.3.7 | | 2.3.9 | 2.3.10 |
| 2.4 | Encryption | | 2.4.1 | | 2.4.3 | | | | | | | |
| 3. | Performance | | | | | | | | | | | |
| 3.1 | Resource Management | | | 3.1.2 | | | 3.1.5 | 3.1.6 | | | | 3.1.10 |
| 3.2 | Traffic Management | | 3.2.1 | 3.2.2 | 3.2.3 | 3.2.4 | 3.2.5 | 3.2.6 | | | 3.2.9 | |
| 4. | Observability | | | | | | | | | | | |
| 4.1 | Monitoring | | 4.1.1 | | 4.1.3 | | 4.1.5 | | | | | |
| 4.2 | Logging | | 4.2.1 | 4.2.2 | 4.2.3 | | 4.2.5 | | | | | |
| 4.3 | Analytics | | | 4.3.2 | 4.3.3 | | | 4.3.6 | 4.3.7 | | 4.3.9 | |
| 5. | Community | | | | | | | | | | | |
| 5.1 | Developer Onboarding | | 5.1.1 | | | 5.1.4 | 5.1.5 | | | 5.1.8 | | |
| 5.2 | Support | | 5.2.1 | | | 5.2.4 | | 5.2.6 | | | | |
| 5.3 | Documentation | | 5.3.1 | | 5.3.3 | | 5.3.5 | | | | | |
| 5.4 | Community Engagement | | 5.4.1 | | 5.4.3 | 5.4.4 | | | 5.4.7 | | 5.4.9 | |
| 5.5 | Portfolio Management | | 5.5.1 | | | 5.5.4 | 5.5.5 | | | | | |
| 6. | Commercial | | | | | | | | | | | |
| 6.1 | Service-Level Agreements | | 6.1.1 | | 6.1.3 | | | 6.1.6 | 6.1.7 | | | |
| 6.2 | Monetization Strategy | | | | | | | 6.2.6 | | 6.2.8 | 6.2.9 | 6.2.10 |
| 6.3 | Account Management | | | 6.3.2 | | | | | 6.3.7 | 6.3.8 | 6.3.9 | |

FIGURE 6.4: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to AFAS' Profit product. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

In order to authenticate consumers of the APIs, tokens are used. Users of a Profit environment are required to request the owner of the environment for a token after which usage of functionalities may commence. In the past, there has been discussion regarding the implementation of the OAuth protocol, but the decision has been made to not invest resources in doing so, considering that the current implementation is fully functional. In terms of threat detection and protection, Profit is very mature considering that it has implemented all practices within this capability. For example, security reviews take place on Profit's internal architecture. Additionally, a third party security company is consulted to perform penetration tests on partner applications. In recent years, such tests have been made to be a standard element of the partner onboarding process.

In the scope of Profit's *Performance*, the product is quite mature with regards to *Resource management* considering that most practices have been implemented, with the exception of *predictive scaling*. Currently, scaling is done in a reactive manner. The practitioners noted that spikes in traffic and amounts of incoming calls are often

tied to small amounts of consumers that are sending faulty requests. In terms of *Traffic Management*, the majority of practices have currently not been implemented. However, this is largely due to deliberate decisions that have been made based on the organization's vision of customer relationships and absence of some functional capabilities that are required to implement these practices. For example, on one hand, consumers do not have the ability to detect which records have changed in the AFAS database, which consequentially means that they are not able to retrieve these specific records. On the other hand, methods with which this could be enabled such as *webhooks*, which are callbacks that can be sent to consumers when an event is triggered such as records being updated (Biehl, 2017), are not implemented. Because of the absence of such methods, consumers are forced to retrieve large volumes of data in some cases, hence Profit has chosen not to implement practices such as *rate limiting*.

With regards to the *Community* surrounding Profit, the majority of practices have been implemented. Concerning the *Developer Onboarding*, consumers are provided with an interactive console with which they can test API behavior and functionalities. Instead of using standards such as OpenAPI, *Documentation* is provided through a separate portal, including code samples and a FAQ section, as well as a Youtube channel which includes video tutorials. Additionally, consumers have the ability to import connectors in bundles through a set of definitions.

In terms of the *Commercial* focus area, a few practices have been implemented for the Profit product. Consumers are to adhere to a SLA, which contains agreements on fair use, time-out policies, and uptime guarantees. However, no custom SLA's are negotiated with individual consumers. Furthermore, no strategy for monetizing the APIs is employed considering that this is already done indirectly through the product's licensing.

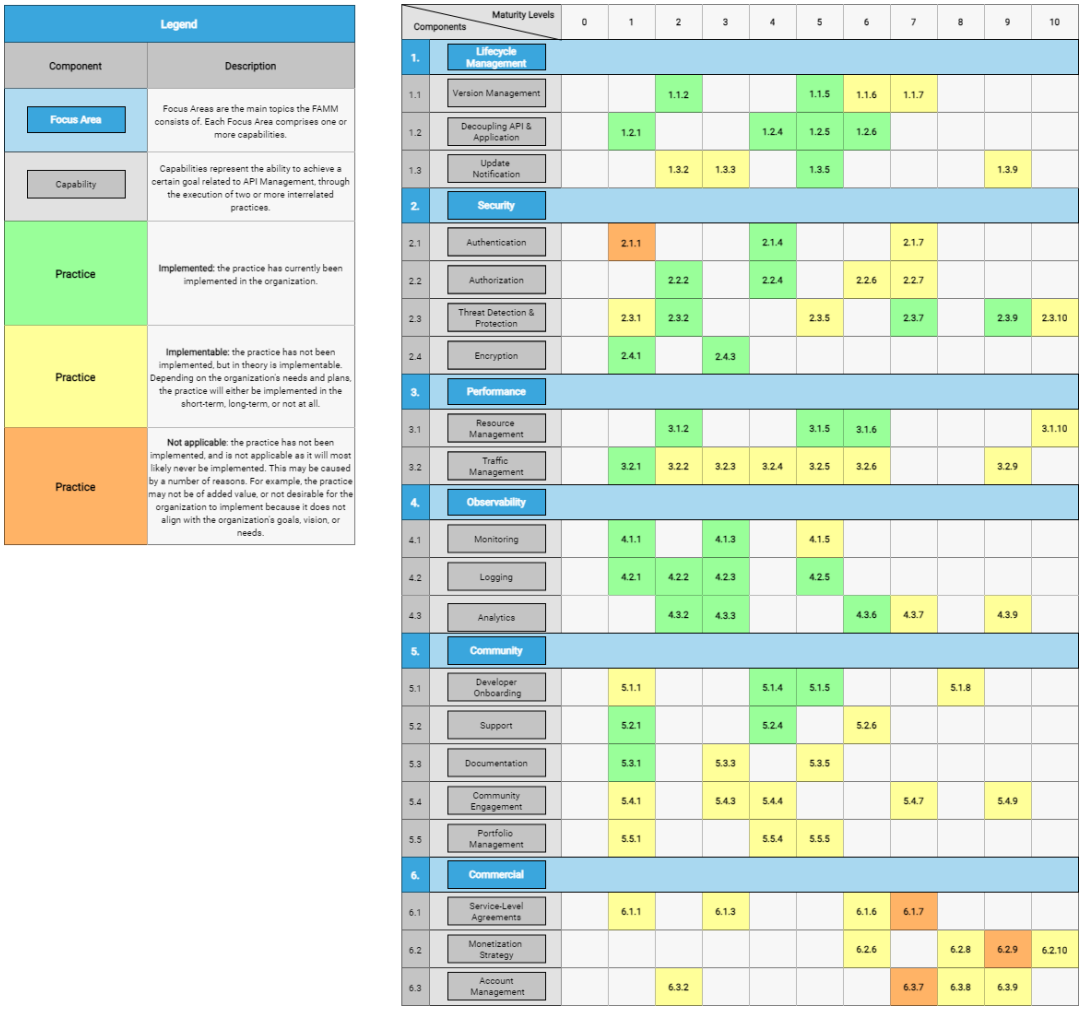
#### AFAS Focus

For AFAS' new product that is currently in development, the API-m-FAMM was applied by one practitioner: the manager of product development. The statuses corresponding to the implementation of each practice as extracted from the spreadsheet provided by practitioners are visualized in Figure 6.5.

With regards to the *Lifecycle Management* focus area, a substantial amount of practices have been implemented. Due to the stage the development of the product is currently in, breaking changes have yet to occur. Hence, an evolutionary versioning strategy is used. However, should this prove to be necessary in the future, measures are already in place to utilize the *maintain multiple API versioning strategy*, which is why it is marked as *implemented*. Currently, consumers are manually informed of any updates of the API since the user base is relatively small as of yet.

In order to improve the *Security* of the product, the OAuth 2.0 protocol has been implemented to authorize consumers. Furthermore, access and refresh tokens are used. With regards to *Threat Detection & Protection*, most practices have already been implemented. Remaining practices such as *DoS protection* will be implemented in the future as the product scales up. Such practices are currently not yet needed, but preparations have been made on a technical level to implement these when needed.

The product's *Performance* is ensured through effective *Resource Management*. In order to do so, Focus has implemented *failover policies* as well as manual scaling. Currently, many *Traffic Management* practices have not yet been implemented. This is a deliberate choice, due to the fact that these are currently not yet needed. The



| Legend | |
|---|---|
| Component | Description |
| Focus Area | Focus Areas are the main topics the FAMM consists of. Each Focus Area comprises one or more capabilities. |
| Capability | Capabilities represent the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices. |
| Practice | **Implemented:** the practice has currently been implemented in the organization. |
| Practice | **Implementable:** the practice has not been implemented, but in theory is implementable. Depending on the organization's needs and plans, the practice will either be implemented in the short-term, long-term, or not at all. |
| Practice | **Not applicable:** the practice has not been implemented, and is not applicable as it will most likely never be implemented. This may be caused by a number of reasons. For example, the practice may not be of added value, or not desirable for the organization to implement because it does not align with the organization's goals, vision, or needs. |

| Components \ Maturity Levels | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Lifecycle Management | | | | | | | | | | | |
| 1.1 | Version Management | | | 1.1.2 | | | 1.1.5 | 1.1.6 | 1.1.7 | | | |
| 1.2 | Decoupling API & Application | | 1.2.1 | | | 1.2.4 | 1.2.5 | 1.2.6 | | | | |
| 1.3 | Update Notification | | | 1.3.2 | 1.3.3 | | 1.3.5 | | | | 1.3.9 | |
| 2. | Security | | | | | | | | | | | |
| 2.1 | Authentication | | 2.1.1 | | | 2.1.4 | | | 2.1.7 | | | |
| 2.2 | Authorization | | | 2.2.2 | | 2.2.4 | | 2.2.6 | 2.2.7 | | | |
| 2.3 | Threat Detection & Protection | | 2.3.1 | 2.3.2 | | | 2.3.5 | | 2.3.7 | | 2.3.9 | 2.3.10 |
| 2.4 | Encryption | | 2.4.1 | | 2.4.3 | | | | | | | |
| 3. | Performance | | | | | | | | | | | |
| 3.1 | Resource Management | | | 3.1.2 | | | 3.1.5 | 3.1.6 | | | | 3.1.10 |
| 3.2 | Traffic Management | | 3.2.1 | 3.2.2 | 3.2.3 | 3.2.4 | 3.2.5 | 3.2.6 | | | 3.2.9 | |
| 4. | Observability | | | | | | | | | | | |
| 4.1 | Monitoring | | 4.1.1 | | 4.1.3 | | 4.1.5 | | | | | |
| 4.2 | Logging | | 4.2.1 | 4.2.2 | 4.2.3 | | 4.2.5 | | | | | |
| 4.3 | Analytics | | | 4.3.2 | 4.3.3 | | | 4.3.6 | 4.3.7 | | 4.3.9 | |
| 5. | Community | | | | | | | | | | | |
| 5.1 | Developer Onboarding | | 5.1.1 | | | 5.1.4 | 5.1.5 | | | 5.1.8 | | |
| 5.2 | Support | | 5.2.1 | | | 5.2.4 | | 5.2.6 | | | | |
| 5.3 | Documentation | | 5.3.1 | | 5.3.3 | | 5.3.5 | | | | | |
| 5.4 | Community Engagement | | 5.4.1 | | 5.4.3 | 5.4.4 | | | 5.4.7 | | 5.4.9 | |
| 5.5 | Portfolio Management | | 5.5.1 | | | 5.5.4 | 5.5.5 | | | | | |
| 6. | Commercial | | | | | | | | | | | |
| 6.1 | Service-Level Agreements | | 6.1.1 | | 6.1.3 | | | 6.1.6 | 6.1.7 | | | |
| 6.2 | Monetization Strategy | | | | | | | 6.2.6 | | 6.2.8 | 6.2.9 | 6.2.10 |
| 6.3 | Account Management | | | 6.3.2 | | | | | 6.3.7 | 6.3.8 | 6.3.9 | |

FIGURE 6.5: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to AFAS' Focus product. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

amount of incoming calls and traffic is relatively low, but preparations have already been made to automate and optimize traffic management in the future when traffic is expected to increase.

Similarly, most practices that are part of the *Community* focus area are not yet relevant for the product. One of the exceptions is the *Developer Onboarding* however, considering that more new consumers are being involved with the product in recent times. Onboarding is achieved through a separate portal which includes a test console, documentation, and the ability to generate SDKs. Doing so is possible because Focus utilizes the OpenAPI specification for the documentation.

Just as is the case for the community surrounding the product, the *Commercial* focus area is largely irrelevant for Focus at this point in time. This is made evident by the absence of implemented practices corresponding to this focus area. However, the practitioner has mentioned that the product is likely to implement similar practices to those that are currently implemented for Profit with regards to the *Commercial* focus area.

### 6.2.2 Multiple Single-Case Study Companies Results

In this section, the results of the companies that participated in the multiple case study are discussed.

#### ConsultComp

In assessing the current maturity of *ConsultComp*, the API-m-FAMM was applied by $Lead\ Engineer_A$, whom was also involved in both evaluation cycles. This was done using the provided spreadsheet. The statuses corresponding to the implementation of each practice contained in the API-m-FAMM are visualized in Figure 6.6.

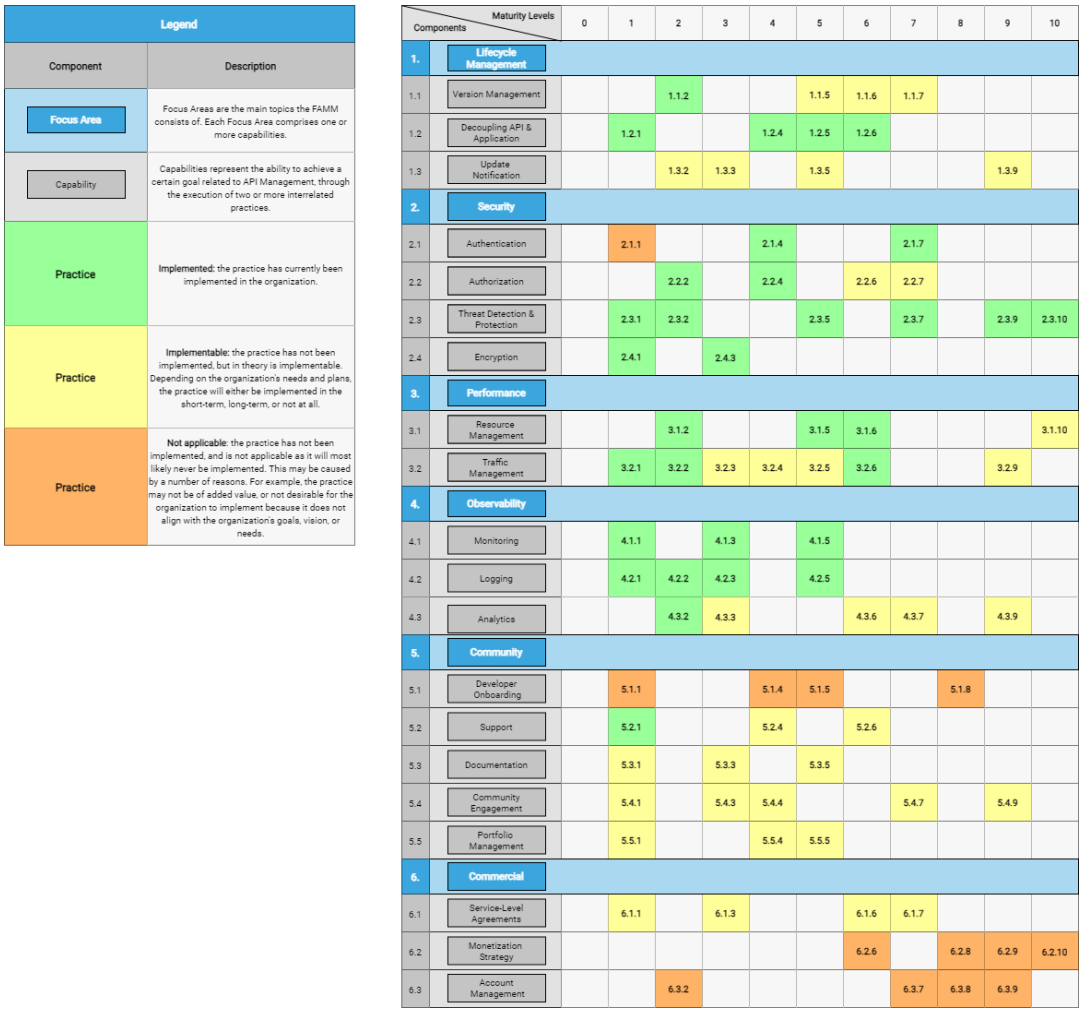



FIGURE 6.6: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to *ConsultComp*. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

In the scope of the *Lifecycle Management* focus area, the internal API is versioned using the *evolutionary versioning strategy*. Furthermore, the API is fully decoupled

from the applications that use it. Considering that the organization's API is exclusively used internally, practices corresponding to *Update Notification* are not implemented. This is due to the fact that when the API is versioned, no mechanisms are needed for notifying external consumers of the update.

Noticeably, a large part of the practices corresponding to the *Security* focus area have been implemented. In order to authenticate internal consumers of the API, an authentication protocol along with the SSO method is used. In terms of *Authorization*, *ConsultComp* is able to perform access control. Furthermore, all practices belonging to *Threat Detection & Protection* and *Encryption* have been implemented to further secure the product.

In terms of *Performance*, scaling and load balancing methods are in place. *Traffic Management* is implemented through practices such as *Time-out Policies* and *Request Caching*. Practices such as rate limiting, throttling and quota management have not been implemented. However, these are marked as *implementable* rather than *not applicable*, considering that the organization is in the process of transitioning towards exposing partner APIs to customers. Hence, the aforementioned practices may need to be implemented in the future.

Regarding real-time *Monitoring* and *Logging* of errors, activity and access, *ConsultComp* is mature, considering that all practices corresponding to these capabilities have been implemented. Most practices in the *Analytics* capability have not been implemented, but this is sensible considering that practices such as *broadcast API status* are only relevant for the case of public or partner APIs.

As is evident when observing 6.6, virtually all practices belonging to the *Community* and *Commercial* focus areas have been marked as *not applicable* or *implementable*. Similarly to capabilities such as *Update Notification* and parts of the *Traffic Management* and *Analytics*, this is logical considering that in the current case of an internal API, there simply is no community surrounding it to manage as of yet.

#### Exact Online

For the case of Exact, the API-m-FAMM was utilized by *Product Manager*, whom was also involved in both evaluation cycles, to assess the current as-is implementation of practices corresponding to the Exact Online product. This was done using the provided spreadsheet. The statuses corresponding to the implementation of each practice contained in the API-m-FAMM are visualized in Figure 6.7.

Interestingly, all practices belonging to the *Lifecycle Management* focus area have been marked as *Implemented*. During interviews that were conducted as part of the previous evaluation cycles, *Product Manager* indicated that currently, an evolutionary versioning strategy is utilized to version Exact Online's APIs. Furthermore, *Product Manager* mentioned that Exact has laid the groundwork for using the multiple versioning strategy to update their REST API to a second version. Hence, both strategies are marked as implemented, as well as the deprecation protocol and backwards compatibility checking practices. The product is also mature with regards to update notification, with the exception of announcing an API versioning schedule. However, *Product Manager* mentioned that implementation of this practice is being worked on, in order to improve developer engagement on Exact's developer portal so that developers are kept up to date with regards to APIs being deprecated.

Similarly, most practices in the *Security* focus area have been marked as implemented. Sensibly, basic authentication was marked as *not applicable* considering that authentication is already implemented in the form of OpenID Connect, which is an

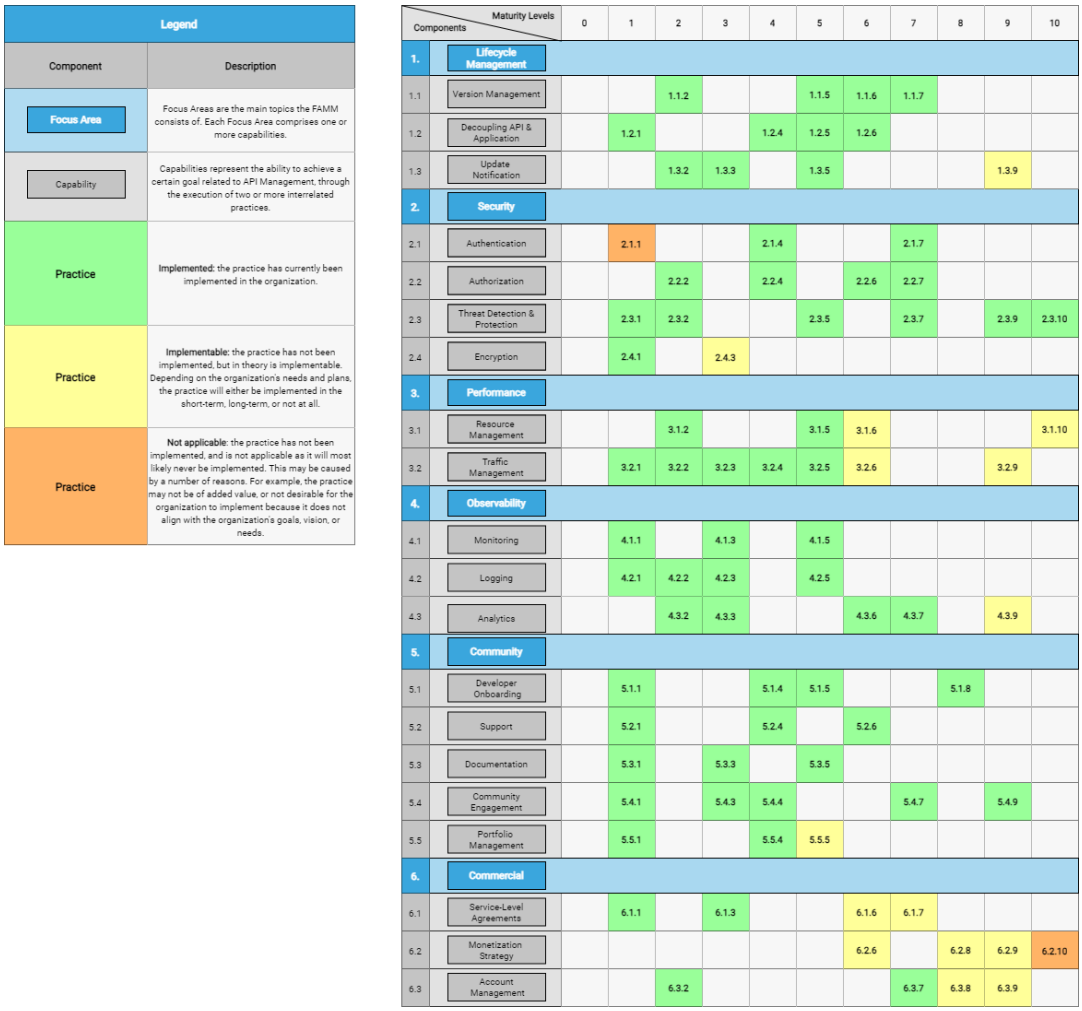



FIGURE 6.7: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to Exact's Online product. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

alternative to basic authentication methods. Furthermore, all authorization practices have been implemented, including OAuth 2.0. This is also the case for threat protection practices, including conducting security reviews: *Product Manager* mentioned that over the course of the past few years, Exact has conducted security audits at 2000 organizations. Furthermore, Exact's Zero Trust Network Architecture is implemented through a third party platform.

In terms of *Observability*, all practices have currently been implemented for Exact Online. The only exception to this is predictive analytics which, similarly to predictive scaling, is a practice that the company is interested in implementing in the future.

Furthermore, Exact is very mature with regards to the *Community* focus area. For example, customers are provided with a sandbox environment, and the company employs a dedicated developer support team as well as a dedicated API evangelist. In fact, one of the experts that was interviewed during the first evaluation cycle, *API Evangelist*, has worked for Exact as an API evangelist for Exact in the past.

Furthermore, Exact provides customers with a developer portal which includes several resources regarding documentation, as well as SDK support for their REST API. Bundling APIs is the only practice that has not been implemented as of yet, but Exact has expressed interest in doing so and considers this practice to be *implementable*.

Lastly, some practices belonging to the *Commercial* focus area have been implemented. Consumers of Exact Online are provided with an extensive SLA, which contains elements such as uptime guarantees, fair use policies, and agreements on rate and data limiting. Furthermore, while access to the product as a whole is monetized through monthly licensing fees, no monetization model that specifically and exclusively applies to the APIs is used.

#### Uber

The API-m-FAMM was used by *Engineer*, whom was involved in the first evaluation cycle, to evaluate the current as-is implementation of practices corresponding to the Uber. This was done using the provided spreadsheet. The statuses corresponding to the implementation of each practice contained in the API-m-FAMM are visualized in Figure 6.8.

In terms of the *Lifecycle Management* focus area, nearly all practices have been implemented. The organization is able to version their APIs using the evolutionary strategy, as well as the multiple API versioning strategy. To this end, Uber also has a deprecation protocol and backwards compatibility checking methods in place. Furthermore, mechanisms to provide consumers with information regarding updates to the API are in place. Additionally, the organization has expressed interest in implementing the *announce versioning schedule* practice.

Regarding the *Security* focus area, all practices have been implemented, signifying that Uber is very mature in this aspect. For example, the *Single Sign-on* authentication method has been implemented, as well as *OAuth authorization scopes* and a *zero trust network access security architecture*.

Similarly, most practices have been implemented for the *Performance* focus area. Considering that Uber is a large-scale multinational organization, this is to be expected. Regarding *Resource Management*, the organization has implemented advanced scaling methods, considering that automatic scaling takes place when certain resource thresholds are exceeded. However, *Engineer* mentioned that there is still room for improvement and preparations are being made to do so, considering that *predictive scaling* is marked as *implementable*. The organization wishes to implement this practice in order to better distribute incoming traffic in different continents and countries by recognizing load patterns ahead of time and scale based on these. Furthermore, Uber has runbooks in place that detail what course of action should be taken when the scaling strategy fails. Additionally, considering that the organization has various data centers that are located in different continents and countries, there are *failover policies* in place which allow the organization to mitigate outages.

Furthermore, Uber is mature with regards to *Observability*, considering the large amount of practices that have been implemented for this focus area. The organization is able to perform all forms of *Monitoring* and *Logging*. Furthermore, the organization wishes to implemented *predictively analytics*, considering that this practice is a requirement for implementing *predictive scaling*.

On the topic of the *Community* surrounding Uber, most practices have been implemented, with a few exceptions. For example, *implement interactive API console* has

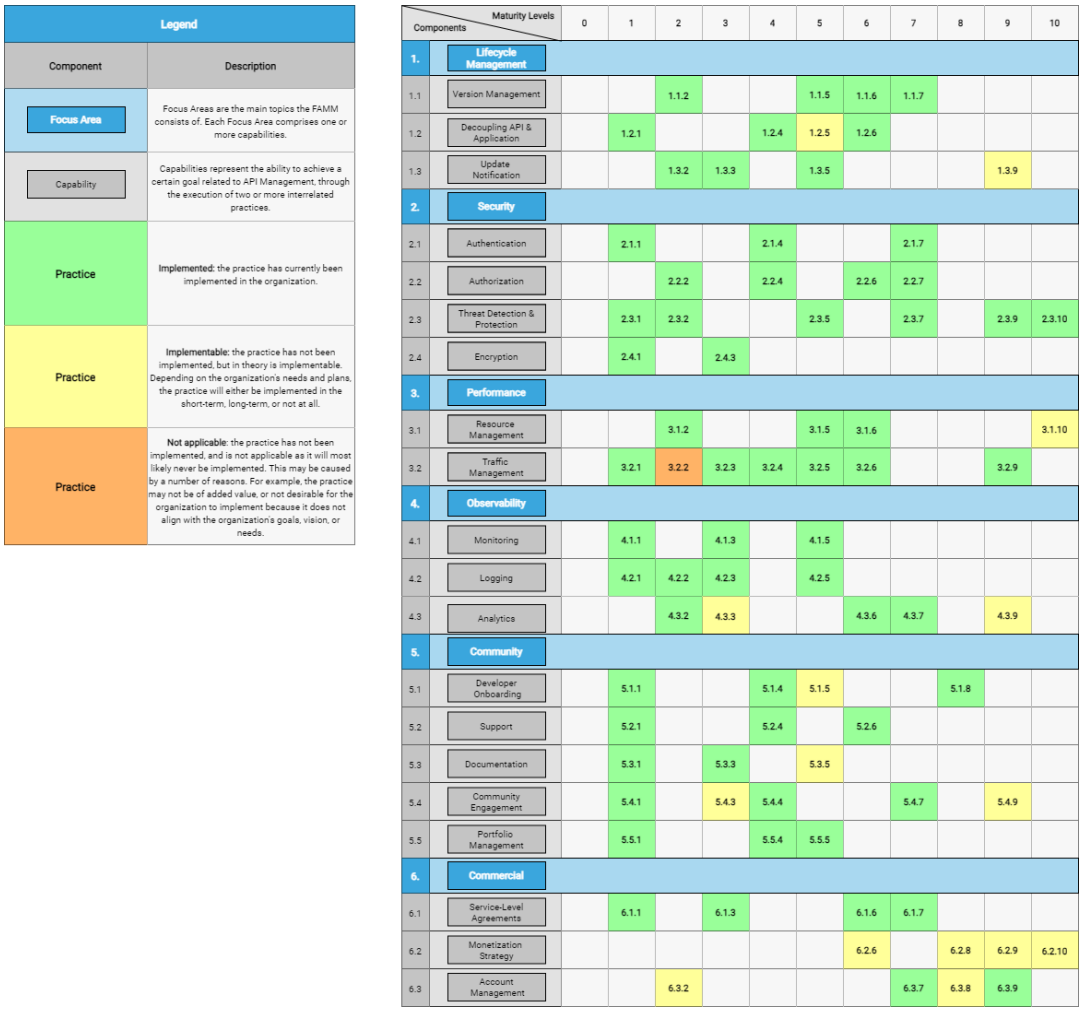



FIGURE 6.8: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to Uber. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

not been marked as implemented. However, an *API sandbox environment* has been implemented as an alternative for this.

Lastly, Uber is quite mature with regards to the *Commercial* focus area. Specifically, this is the case for the *Service-Level Agreements* capability, considering that the organization is able to proactively monitor their SLA as well as provide consumers with customized SLAs if necessary.

### 6.2.3 Misinterpreted Case Study Company Results

In this subsection, the results of the case study at *EAMComp* are discussed. However, considering that the case study protocol was misinterpreted by the practitioner, these results are discussed separately from the other results of the multiple case studies.

### EAMComp

In order to assess the maturity of *EAMComp*, the API-m-FAMM was applied by a practitioner who has not been previously involved in any evaluation cycles. This was done using the provided spreadsheet. The statuses corresponding to the implementation of each practice contained in the API-m-FAMM are visualized in Figure 6.9.

| Legend | |
|---|---|
| **Component** | **Description** |
| Focus Area | Focus Areas are the main topics the FAMM consists of. Each Focus Area comprises one or more capabilities. |
| Capability | Capabilities represent the ability to achieve a certain goal related to API Management, through the execution of two or more interrelated practices. |
| Practice | **Implemented:** the practice has currently been implemented in the organization. |
| Practice | **Implementable:** the practice has not been implemented, but in theory is implementable. Depending on the organization's needs and plans, the practice will either be implemented in the short-term, long-term, or not at all. |
| Practice | **Not applicable:** the practice has not been implemented, and is not applicable as it will most likely never be implemented. This may be caused by a number of reasons. For example, the practice may not be of added value, or not desirable for the organization to implement because it does not align with the organization's goals, vision, or needs. |

| Components \ Maturity Levels | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Lifecycle Management | | | | | | | | | | | |
| 1.1 | Version Management | | | 1.1.2 | | | 1.1.5 | 1.1.6 | 1.1.7 | | | |
| 1.2 | Decoupling API & Application | | 1.2.1 | | | 1.2.4 | 1.2.5 | 1.2.6 | | | | |
| 1.3 | Update Notification | | | 1.3.2 | 1.3.3 | | 1.3.5 | | | | 1.3.9 | |
| 2. | Security | | | | | | | | | | | |
| 2.1 | Authentication | | 2.1.1 | | | 2.1.4 | | | 2.1.7 | | | |
| 2.2 | Authorization | | | 2.2.2 | | 2.2.4 | | 2.2.6 | 2.2.7 | | | |
| 2.3 | Threat Detection & Protection | | 2.3.1 | 2.3.2 | | | 2.3.5 | | 2.3.7 | | 2.3.9 | 2.3.10 |
| 2.4 | Encryption | | 2.4.1 | 2.4.2 | 2.4.3 | | | | | | | |
| 3. | Performance | | | | | | | | | | | |
| 3.1 | Resource Management | | | 3.1.2 | | | 3.1.5 | 3.1.6 | | | | 3.1.10 |
| 3.2 | Traffic Management | | 3.2.1 | 3.2.2 | 3.2.3 | 3.2.4 | 3.2.5 | 3.2.6 | | | 3.2.9 | |
| 4. | Observability | | | | | | | | | | | |
| 4.1 | Monitoring | | 4.1.1 | | 4.1.3 | | 4.1.5 | | | | | |
| 4.2 | Logging | | 4.2.1 | 4.2.2 | 4.2.3 | | 4.2.5 | | | | | |
| 4.3 | Analytics | | | 4.3.2 | 4.3.3 | | | 4.3.6 | 4.3.7 | | 4.3.9 | |
| 5. | Community | | | | | | | | | | | |
| 5.1 | Developer Onboarding | | 5.1.1 | | | 5.1.4 | 5.1.5 | | | 5.1.8 | | |
| 5.2 | Support | | 5.2.1 | | | 5.2.4 | | 5.2.6 | | | | |
| 5.3 | Documentation | | 5.3.1 | | 5.3.3 | | 5.3.5 | | | | | |
| 5.4 | Community Engagement | | 5.4.1 | | 5.4.3 | 5.4.4 | | | 5.4.7 | | 5.4.9 | |
| 5.5 | Portfolio Management | | 5.5.1 | | | 5.5.4 | 5.5.5 | | | | | |
| 6. | Commercial | | | | | | | | | | | |
| 6.1 | Service-Level Agreements | | 6.1.1 | | 6.1.3 | | | 6.1.6 | 6.1.7 | | | |
| 6.2 | Monetization Strategy | | | | | | | 6.2.6 | | 6.2.8 | 6.2.9 | 6.2.10 |
| 6.3 | Account Management | | | 6.3.2 | | | | | 6.3.7 | 6.3.8 | 6.3.9 | |

FIGURE 6.9: The API-m-FAMM as applied to the current as-is implementation of practices corresponding to *EAMComp*. Please consult the legend on the top left-hand side of the figure for more information regarding the manner in which various colors corresponding to practices should be interpreted.

As may be easily observed when examining Figure 6.9, the answers provided through the spreadsheet differ from the results obtained from other case studies. Interestingly, only one practice per capability is marked using the various implementation categories. Because of this, the researchers hypothesize that the practitioner misinterpreted one of the instructions as part of the case study protocol that case study participants were provided with. While participants were asked to denote the implementation status for each practice, it seems as though the practitioner has denoted the implementation status for each practice of the highest maturity within a

capability and that he is knowledgeable on. The researchers suspect that this was caused due to the fact that the practitioner did not participate in previous evaluation cycles, or because the case study protocol was not phrased clearly enough. Unfortunately, since the practitioner did not respond to the researcher's request for a follow-up discussion of the provided results, this suspicious has not been able to be confirmed.

As a consequence of the limited answers that were provided and uncertainty as to how the case study protocol was interpreted, the researchers are not able to accurately assess *EAMComp*'s API management maturity. Regardless, it was decided to include this case in this work in order to highlight the varying ways the API-mFAMM may be utilized by practitioners.

## 6.3 Discussion of Results

Now that the results of the individual results of the case companies have been discussed, the overall findings of the case study are analyzed in this section. First, the various organizations and products that were evaluated are benchmarked in Table 6.1, based on which observations are made. Next, the results of the evaluative survey, as shown in Table 6.2, that was filled in by practitioners as part of the case study protocol are discussed.

All results that were extracted from the spreadsheets that the participating practitioners have filled out and subsequently returned to the researchers have been summarized in Table 6.1. However, results provided by *EAMComp* have been excluded from this table, as there was no data provided regarding the majority of practices. Table 6.1 shows the amount of practices that are *implemented* for each capability. Consequently, this means that the other implementation categories (*implementable* and *not applicable*) are not taken into account in this overview. Furthermore, before discussing these results, it should be noted that the percentages representing the average amount of practices that are implemented in each focus area should be interpreted in an indicative manner, as some practices may encompass a much larger amount of work than others. Moreover, some case companies vary heavily in terms of their vision, size, usage of APIs, and goals. For example: Uber, AFAS Profit and Exact may be roughly classified as large and mature organizations that have been exposing public or partner APIs for a long period of time, which is reflected by the observation that these organizations have implemented the majority of practices for most focus areas. *ConsultComp* is also a large organization in size, but the product that was evaluated has only been utilizing internal APIs for a short period of time. Hence, some focus areas are largely irrelevant within the scope of this product. Similarly, the AFAS Focus product has been utilizing partner APIs for a relatively short period of time and is currently in the process of expanding the services it provides as well as the assets it exposes. Due to the current scale of the product and the stage of development it is in, some practices are currently not yet necessary to be implemented.

These differences are reflected by these organizations' maturity in API management across the various focus areas, as will be discussed in the following paragraphs. First however, we make several observations that were made as a result of examining and comparing the visual representations of the spreadsheets that were filled out by practitioners, as shown in the previous section.

TABLE 6.1: The results of the deployment of the API-m-FAMM at the case companies. For each capability, the amount of implemented practices is shown. The percentages indicate the share of practices that have been implemented out of the total amount of practices that are assigned to the focus area. All percentages are colored, with those approaching 100% being colored green, and those approaching 0% being colored orange. Please note that these percentages should be interpreted in an indicative manner, as the practices are not weighted, and some practices are more extensive than others. Furthermore, results provided by *EAMComp* have been omitted from this table.

| | **Focus Area** | **AFAS Focus** | **AFAS Profit** | *ConsultComp* | **Exact Online** | **Uber** | **Avg** | **Stdev** |
|---|---|---|---|---|---|---|---|---|
| **1** | **Lifecycle Management** | 58% | 50% | 42% | 92% | 83% | 65% | |
| 1.1 | Version Management | 2 | 2 | 1 | 4 | 4 | 2.6 | 1.20 |
| 1.2 | Decoupling API & Application | 4 | 2 | 4 | 4 | 3 | 3.4 | 0.80 |
| 1.3 | Update Notification | 1 | 2 | 0 | 3 | 3 | 1.8 | 1.17 |
| **2** | **Security** | 53% | 73% | 80% | 87% | 100% | 79% | |
| 2.1 | Authentication | 1 | 1 | 2 | 2 | 3 | 1.8 | 0.75 |
| 2.2 | Authorization | 2 | 2 | 2 | 4 | 4 | 2.6 | 0.98 |
| 2.3 | Threat Detection & Protection | 3 | 6 | 6 | 6 | 6 | 5.4 | 1.20 |
| 2.4 | Encryption | 2 | 2 | 2 | 1 | 2 | 1.8 | 0.40 |
| **3** | **Performance** | 36% | 45% | 55% | 64% | 82% | 56% | |
| 3.1 | Resource Management | 3 | 3 | 3 | 2 | 3 | 2.8 | 0.40 |
| 3.2 | Traffic Management | 1 | 2 | 3 | 5 | 6 | 3.4 | 1.86 |
| **4** | **Observability** | 75% | 67% | 67% | 92% | 83% | 77% | |
| 4.1 | Monitoring | 2 | 3 | 3 | 3 | 3 | 2.8 | 0.40 |
| 4.2 | Logging | 4 | 3 | 4 | 4 | 4 | 3.8 | 0.40 |
| 4.3 | Analytics | 3 | 2 | 1 | 4 | 3 | 2.6 | 1.02 |
| **5** | **Community** | 28% | 78% | 6% | 94% | 78% | 57% | |
| 5.1 | Developer Onboarding | 2 | 1 | 0 | 4 | 3 | 2.0 | 1.41 |
| 5.2 | Support | 2 | 3 | 1 | 3 | 3 | 2.4 | 0.80 |
| 5.3 | Documentation | 1 | 2 | 0 | 3 | 2 | 1.6 | 1.02 |
| 5.4 | Community Engagement | 0 | 5 | 0 | 5 | 3 | 2.6 | 2.24 |
| 5.5 | Portfolio Management | 0 | 3 | 0 | 2 | 3 | 1.6 | 1.36 |
| **6** | **Commercial** | 0% | 42% | 0% | 33% | 50% | 25% | |
| 6.1 | Service-Level Agreements | 0 | 2 | 0 | 2 | 4 | 1.6 | 1.50 |
| 6.2 | Monetization Strategy | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 |
| 6.3 | Account Management | 0 | 3 | 0 | 2 | 2 | 1.4 | 1.20 |

Firstly, it was observed that some practices are mutually exclusive. This phenomenon may be observed in the *Authentication* capability, where in some cases the *Implement Basic Authentication* has been marked as *not applicable* when the *Implement Authentication Protocol* had been marked as *implemented*. This is indeed a logical result, considering that only one method of authentication is necessary. This means that when an organization has implemented an authentication protocol such as OpenID Connect, the implementation of HTTP basic authentication is no longer applicable. Hence, a choice has to be made between these two practices, resulting in the practice of lower maturity having to be marked as *not applicable* or *implementable* when the practice of higher maturity is marked as *implemented*. Similarly, such choices are also observed between the *Implement Interactive API Console* and *Provide Sandbox Environment Support* practices. Likewise, it is likely that this phenomena would have also been observed for the practices in the *Monetization Strategy* (if their implementation had been observed as part of this case study), as only one monetization strategy is needed to monetize an organization's APIs.

Secondly, it may be observed that the mature organizations (Uber, Exact and AFAS Profit) as a whole have implemented the majority of practices across most focus areas. This is particularly the case for *Lifecycle Management*, *Security*, *Performance*, *Observability* and *Community*. However, the AFAS Profit product forms an exception to this observation, considering the relatively high amount of practices that are *not applicable* and *implementable*. The researchers attribute this to the product being fully developed. The reason for this is that the practitioners noted that practices that have been marked as such have been discussed internally in the past, and were concluded to not be desirable to be implemented due to them not aligning with the prevalent vision and goal of the product. Additionally, as is discussed in the previous section, some practices are not able to be implemented due to technical limitations, or are decided not be implemented due to time-priority trade-offs.

Thirdly, we observe that within the earlier mentioned focus areas, some practices on the higher end in terms of maturity are often not implemented across the board, such as *Announce Versioning Roadmap*, *Implement Predictive Scaling*, and *Prioritize Traffic*. We interpret this as supportive evidence for having assigned these practices to high maturity levels. Furthermore, during the discussions that were had as part of the evaluation cycles and the single embedded single-case study, several organizations expressed interest in implementing these practices in the future. It should be noted however, that the time-span in which organizations plan on implementing such practices may vary greatly.

Fourthly, it may be observed that some focus areas and capabilities seem to be less important for AFAS Focus and *ConsultComp*, judging from the amount of practices that have been marked as *implementable* and *not applicable*. In particular, this is the case for the *Update Notification* and *Traffic Management* capabilities, and the *Community* and *Commercial* focus areas. However, this observed lack of implementation of practices for these products may be attributed to different factors.

In the case of AFAS Focus, this is caused by the fact the product is currently still in development. As a result, for example, the product has not had to version its APIs in major ways as of yet and thus has not needed to notify consumers of this, in addition to the community surrounding the product being it its infancy. However, some practices are implemented to be able to onboard developers, as well as to provide

support. Furthermore, due to the low amount of current consumers, there is no need to impose rate or data limits.

For the case of *ConsultComp*, the lack of implementation of practices in the aforementioned capabilities and focus areas may be attributed to the fact that the organization exclusively utilizes internal APIs. Since a community is non-existant, there are no developers to onboard, which is why these practices have been marked as *not applicable* and the rest of the practices in this focus area are *implementable*. The researchers suspect that similar implementation patterns may be observed in other organizations and products that are in the process of developing and expanding their product and its services, as well as those that exclusively use APIs internally. These differences in implementation patterns between the mature organizations and the ones that are developing or utilizing internal APIs is also made evident when examining the standard deviations of the capabilities and focus areas that were mentioned in the previous paragraph, as can be seen in Table 6.1.

Lastly, a common trend that may be observed across all organizations is that the *Commercial* focus area is underdeveloped. Particularly, this is the case for the *Monetization Strategy* capability, considering that no organization has marked any of these practices as *implemented*. However, the practitioners at Exact and *EAMComp* have expressed interest in implementing a monetization strategy in the future. Specifically, the practitioner at Exact mentioned that he expects that monetization for APIs will become increasingly more common in the future, and that he suspects that large-scale organizations that expose high-traffic public APIs already monetize their APIs. However, no concrete evidence of this has been found as part of this case study and its participants.

In addition to the observations listed above, some limitations were uncovered through the discussions that were conducted as part of the embedded single case study. Especially with regards to the *Commercial* focus area, it is likely that the practitioner applying the API-m-FAMM is not knowledgeable on the long-term vision and strategy of the organization. While the practitioner may often be able to assess whether a practice is implementable on a technical level, he or she is not able to assess whether the practice will actually be implemented in the future. In such cases, it was found that the practitioner is likely to mark a practice as *implementable*, rather than *not applicable*. Furthermore, in some cases practitioners expressed that they were unsure as to the way some practices should be interpreted, which often resulted in such practices being interpreted in a different way than was intended by the researchers. For example, some practitioners had initially marked one of the monetization strategy practices as *implemented* because they thought these practices applied to the monetization of the product as a whole, instead of the APIs themselves. This was also the case in assessing whether the organization's SLA should be considered as informal as opposed to formal or strict, as is mentioned in the *Publish Informal SLA* and *Provide SLA* practices, as well as manual versus automatic and predictive scaling as part of the *Implement Scaling* and *Implement Predictive Scaling* practices. Moreover, in some cases the dependencies that are present between practices were not taken into consideration when checking the implementation of practices. Because of these observations, we conclude that in order for researchers to obtain accurate results of an organization's current situation with regards to API management practices, a follow-up discussion should be conducted with practitioners to weed out such inconsistencies and differences in interpretation.

In spite of these limitations however, we have found that, generally speaking, the practitioners experienced the usage of the API-m-FAMM in a positive manner. As can be seen in Table 6.2, the *ease of use*, *usefulness*, and *effectiveness*, were all ranked with an average score of 4 or higher. The researchers interpret this as evidence for the usefulness and effectiveness of the API-m-FAMM in achieving its goal of aiding organizations in assessing their current maturity in API management and guiding them towards achieving higher levels of maturity. However, the average score attributed to the API-m-FAMM's operational feasibility is notably lower when compared to the other criteria. In particular, the scores given by practitioners corresponding to the AFAS Focus and Profit products are among the lowest. This may be explained by the fact that the AFAS Profit product is almost fully matured and developed, which reduces the incentive among its practitioners to repeatedly consult the API-m-FAMM for guidance. This is further compounded due to the practitioners already being aware of most practices that have not yet been implemented, as well as having previously discussed them internally in the past. For the case of AFAS Focus, this may be explained by the observation that a large portion of focus areas and capabilities are irrelevant for the product given its current development stage, which in turn reduces the effectiveness of the API-m-FAMM in this case.

TABLE 6.2: The rankings given by the case companies' practitioners in response to the questions corresponding to the four evaluation criteria, as defined in Section 6.1.3, as well as their averages.

| Interviewee | Operational Feasibility | Ease of Use | Usefulness | Effectiveness |
|---|---|---|---|---|
| AFAS Profit[1] | 2 | 4 | 4 | 3.5 |
| AFAS Focus | 2 | 3 | 4 | 3 |
| *ConsultComp* | 4 | 4 | 3 | 4 |
| Exact Online | 4 | 5 | 5 | 4 |
| Uber | 3 | 4 | 4 | 5 |
| *EAMComp* | 4 | 4 | 4 | 5 |
| **Average** | 3.2 | 4.0 | 4.0 | 4.1 |

[1] Scores for this product were provided by two practitioners. As such, the average of these scores is shown.

Interestingly, the API-m-FAMM was ranked fairly high by *ConsultComp*'s practitioner, regardless of the fact that, similarly to the AFAS Focus product, some focus areas and capabilities do not apply to the product given its utilization of internal APIs. However, *ConsultComp*'s intent of developing partner APIs in the future may provide an explanation for the practitioner's appreciation of the API-m-FAMM. Furthermore, it is interesting to note that even though the case study protocol was misinterpreted by *EAMComp*'s practitioner, the API-m-FAMM was nevertheless ranked positively. Because of this, it seems that even though the misinterpretation of the case study protocol was inconvenient for the researchers in terms of data extraction and assessing the organization's level of maturity, this did not negatively impact the API-m-FAMM in achieving its intended goal. Regarding the average score given to the *operational feasibility* criterion, the researchers suspect that this is on the low end due to a lack of incentive for practitioners to re-use the API-m-FAMM after the initial assessment.

## 6.4 Changes Made as Result of Case Study Findings (API-m-FAMM v3.0)

As a result of findings from the case studies that are described in this chapter, several changes were made to the API-m-FAMM. The decisions to make these changes are based on the discussions that were held as part of the embedded single-case study as well as general observations that were made as a result of analyzing the results of all case studies.

First, the decision is made to merge two of the implementation categories of practices that are described in Section 6.1.3: *not implementable* and *not applicable*. These categories were merged into the **not applicable** category. During the discussion sessions, it became clear that these two categories were used interchangeably, and their difference caused confusion among practitioners. After inquiring as to what caused this confusion, the practitioners noted that given an investment of sufficient time and resources, every practice is implementable in theory. However, the practices that were initially marked as *not implementable* were actually deemed to be undesirable to implement due to the fact that they do not fit the organization's scope, needs, vision, or goals, rather than technically not being implementable.

For example, in the case of AFAS Profit, the *multiple API versioning strategy* was initially marked as *not implementable*. However, the practitioners noted that this strategy has yet to become necessary in versioning the APIs due to the fact that the version of the connectors are directly tied to the version of the database on the backend. As such, it was determined that for AFAS Profit, this practice is *not applicable* rather than *not implementable*. Because this seemed to be the case for all practices that were initially denoted by the *not implementable* category, it was rendered obsolete, which is also further supported by the limited observation of practices being marked as such. The description of the *not applicable* category was then extended to the following: *"the practice has not been implemented, and is not applicable as it will most likely never be implemented. This may be caused by a number of reasons. For example, the practice may not be of added value, or not desirable for the organization to implement because it does not align with the organization's goals, vision, or needs."*.

Secondly, a similar decision was made to merge the *implementable but not planned* and *implementable and planned* categories into an implementation category named **implementable**. This was done due to a number of reasons. Firstly, during the discussion sessions, it became clear that the description corresponding to the *implementable and planned* was widely interpretable for practitioners. Specifically, the phrasing: *"planned for implementation in the **foreseeable future**"* caused confusion, considering that the time frame this might apply to was not explicitly specified. For example, the practitioner that was interviewed during the case study with AFAS Focus had initially marked the *predictive analytics* practice as *implementable and planned*. However, when discussing this, it was mentioned that this was done solely based on a past expression of interest towards implementing this practice during an internal discussion session. Moreover, the practice had not yet been placed on any formal development roadmap. In addition to this example, multiple cases were encountered where the practitioners of other case companies mentioned being unsure as to when a practice that they had initially marked as *implementable and planned* would precisely be implemented, regardless of the fact that it had already been placed on a development roadmap. Considering that long-term roadmaps are often dynamic

and priorities tend to change over time, in addition to the complexity involved with implementing some practices (especially those of higher maturity), it is generally difficult for practitioners to accurately estimate when practices will be implemented.

As such, it was decided that the confusion and ambiguity caused by the *implementable and planned* category may best be avoided by excluding any aspects of time estimation and prioritization from the API-m-FAMM. While the API-m-FAMM may assist practitioners in internal discussions aimed at prioritizing and planning the implementation of practices, its principal goal is to assess the current degree of maturity of an organization's current state in API management. Hence, for the purpose of this research and the data collection for this case study, processes such as planning and placing practices on development roadmaps are excluded by merging the *implementable but not planned* and *implementable and planned* categories into the **implementable** category. This implementation category is described as: *the practice has not been implemented, but in theory is implementable. Depending on the organization's needs and plans, the practice will either be implemented in the short-term, long-term, or not at all.*

Furthermore, some additional changes are made to practices as a result of the discussion sessions with practitioners. One practice was removed altogether, and the descriptions of six practices were modified. Specifically, the following changes were made to the following practices:

- **Perform Request Rate Limiting**: this practice was extended to also comprise error limiting. In the case of AFAS Profit, this is implemented by placing consumers on a temporary denylist when they perform an excessive amount of faulty calls within a predefined time span.
- **Prevent Sensitive Data Exposure**: this practice was removed. During discussions, this practice caused confusion due to the observation that this practice is already captured by the *Implement Transport Layer Encryption* and *Decouple Internal & External Data Model* practices. Additionally, after further investigation this practice was deemed to be out of scope, considering that the scope of this practice involves app data storage in general, as opposed to API management.
- **Implement Predictive Scaling**: the description of this practice was modified. Originally, the description mentioned that this practice may be implemented 'manually or automatically', which caused confusion due to the fact that these methods are already capture in the *Implement Scaling* practice. Because predictive scaling is envisioned by practitioners and the researchers to be done automatically, the manual element was removed from the description.
- **Monitor Resource Usage**: the description of this practice was expanded. During discussions, it became clear that monitoring resources does not always specifically involve metrics such as CPU and disk usage. Instead, rough approximations may be used to determine resource usage instead, which is why the description was expanded to clarify this.

In addition to these changes, a small number of changes were made as a result of practitioners identifying errors such as typos. These were corrected, after which the document describing the API-m-FAMM's source data was updated accordingly (Mathijssen, Overeem, & Jansen, 2021b). Furthermore, the visual variant of the API-m-FAMM was updated to incorporate the removal of the *prevent sensitive data exposure* practice. This final version of the model, API-m-FAMM v3.0, can be seen in Figure 6.10.

| Components \ Maturity Levels | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1. Lifecycle Management** | | | | | | | | | | | |
| 1.1. Version Management | | | Implement Evolutionary API Strategy | | | Implement Multiple API Versions Strategy | Implement API Deprecation Protocol | Check Backwards Compatibility | | | |
| 1.2. Decoupling API & Application | | Decouple API & Software Versioning | | | Decouple Internal & External Data Model | Decouple Internal & External Data Format | Decouple Internal & External Transport Protocol | | | | |
| 1.3. Update Notification | | | Distribute Changelogs | Distribute Versioning Notification through Channel(s) | | Extend API with Versioning Information | | | | Announce API Versioning Roadmap | |
| **2. Security** | | | | | | | | | | | |
| 2.1. Authentication | | Implement Basic Authentication | | | Implement Authentication Protocol | | | Implement Single Sign-on | | | |
| 2.2. Authorization | | | Implement Access Control | | Implement Token Management | | Implement Standardized Authorization Protocol | Implement Authorization Scopes | | | |
| 2.3. Threat Detection & Protection | | Implement Allow & Deny IP Address Lists | Implement Injection Threat Protection Policies | | | Implement DoS Protection | | Implement Security Breach Protocol | | Conduct Security Review | Implement Zero Trust Network Access |
| 2.4. Encryption | | Implement Transport Layer Encryption | | Implement Certificate Management | | | | | | | |
| **3. Performance** | | | | | | | | | | | |
| 3.1. Resource management | | | Implement Load Balancing | | | Implement Scaling | Implement Failover Policies | | | | Implement Predictive Scaling |
| 3.2. Traffic Management | | Set Timeout Policies | Implement Request Caching | Perform Request Rate Limiting | Perform Request Rate Throttling | Manage Quota | Apply Data Volume Limits | | | Prioritize Traffic | |
| **4. Observability** | | | | | | | | | | | |
| 4.1. Monitoring | | Monitor API Health | | Monitor API Performance | | Monitor Resource Usage | | | | | |
| 4.2. Logging | | Log Errors | Log Access Attempts | Log Activity | | Audit User Activity | | | | | |
| 4.3 Analytics | | | Report Errors | Broadcast API Status | | | Generate Custom Analysis Reports | Set Alerts | | Enable Predictive Analytics | |
| **5. Community** | | | | | | | | | | | |
| 5.1. Developer Onboarding | | Facilitate Developer Registration | | | Provide SDK Support | Implement Interactive API Console | | | Provide Sandbox Environment | | |
| 5.2. Support | | Establish Communication Channel | | | Manage Support Issues | | Dedicate Developer Support Team | | | | |
| 5.3. Documentation | | Use Standard for Reference Documentation | | Provide Start-up Documentation Including Samples | | Create Video Tutorials | | | | | |
| 5.4. Community Engagement | | Maintain Social Media Presence | | Provide Community Forum | Provide Developer Portal | | | Organize Events | | Dedicate API Evangelist | |
| 5.5. Portfolio Management | | Enable API Discovery | | | Provide API Catalog | Bundle APIs | | | | | |
| **6. Commercial** | | | | | | | | | | | |
| 6.1. Service-Level Agreements | | Publish Informal SLA | | Provide SLA | | | Monitor SLA Proactively | Customize Personalized SLA | | | |
| 6.2. Monetization Strategy | | | | | | | Adopt Subscription-Based Monetization Model | | Adopt Tier-Based Monetization Model | Adopt Freemium Monetization Model | Adopt Metering-Based Monetization Model |
| 6.3. Account Management | | | Implement Subscription Management System | | | | | Report on API Program Business Value | Provide Subscription Report to Customer | Proactively Suggest Optimizations to Customer | |

FIGURE 6.10: API-m-FAMM v3.0. This is the final version of the API-m-FAMM, which incorporates the changes that are applied as a result of the conducted case studies.

# Chapter 7

# Discussion

Considering the results from the experts interviews and case studies have been discussed in previous chapters, this chapter is aimed at first reflecting on the design and creation process of the API-m-FAMM. Next, scientific contributions made as part of this study are presented. Then, all threats to the validity of this research are identified, along with the actions that were taken to mitigate them. Finally, limitations as well as opportunities and possibilities for future work are discussed.

## 7.1 Findings and Implications

While the final version of the API-m-FAMM is structured in an effective and understandable manner, in order to arrive at this point many extensive discussions were held among the researchers in the early stages of scoping, structuring, and populating the model. The difficulty of decisions that had to be made were often tied to the fact that, as a research subject, the topic of API management is often relatively broad, commercialized, and abstract. When first investigating the topic, the variety of sub-topics that were encountered and that are tied to API management was often quite overwhelming. A challenge that resulted from this that the researchers were faced with throughout this study lied in determining the correct level of abstraction. For example, topics such as microservices were encountered, which is an architectural style for web applications consisting of small web services that often use APIs to communicate between themselves. It could be argued that such a topic, as well as corresponding sub-topics such as server- and client service discovery, should be included in the API-m-FAMM considering that APIs play a role in it.

Similarly, such scoping dilemmas were encountered with deciding whether architectural components such as gateways should be included in the model. However, considering that capabilities such as traffic and resource management may be achieved without using a gateway or alternatives such as proxies, and these approaches each have (dis)advantages without necessarily being more mature than the other, they were excluded from the model's scope. Another challenge was encountered in deciding whether the design and creation of APIs should be included in the model, considering that it is the first of the stages the API lifecycle (as shown in Figure 3.5) consists of. Inclusion of this stage would entail that various design methodologies such as the API first approach (Rivero, Heil, Grigera, Gaedke, & Rossi, 2013), which advocates for designing the API's contract first before writing any code, should either be included or excluded. Inclusion would result in such practices becoming too abstract in comparison to those assigned to other capabilities and focus areas, while exclusion would result in practices being incomplete. Hence, the decision was made to exclude all design-related practices from the scope. The main solution that was used to solve these scoping dilemmas was to identify what (sub)topics belong to the core of API management. This was determined through

expert interviews, based on which the scope of the model was adjusted throughout the process.

These scoping adjustments are made evident by analyzing the distribution of the amounts of practices and capabilities throughout the API-m-FAMM's evolution. As can be seen in Table 7.1, the number of practices and capabilities that were included in the model declined throughout the iterations. The decline that is observed between the collection of practices and capabilities that were extracted through the SLR and the first version was mostly caused due to the fact that, in hindsight, a large part of these were classified incorrectly due to a lack of knowledge of the topic in this stage of the research. For example, many processes that ended up being classified as practices in the final version of the model were initially marked as capabilities, and many processes that initially were marked as practices were later deemed too granular and were ultimately merged into more substantial practices. This was done based on an increasing body of knowledge that gradually grew over the course of this research, by using information gathered from (grey) literature, expert interviews, and discussion sessions. This decline may also be attributed to scoping decisions that were described throughout this work, as well as those outlined in the previous paragraph.

TABLE 7.1: Number of practices, capabilities, and focus areas that were extracted by the SLR, and subsequently comprised by the three major versions of the API-m-FAMM: v1.0, v2.0, and v3.0.

| Component | SLR | API-m-FAMM v1.0 | API-m-FAMM v2.0 | API-m-FAMM v3.0 |
|---|---|---|---|---|
| **Practice** | 114 | 87 | 81 | 80 |
| **Capability** | 39 | 23 | 20 | 20 |
| **Focus Area** | 0 | 6 | 6 | 6 |

Another point of discussion is that of the heterogeneous distribution among practices that that may be observed across all maturity levels of the API-m-FAMM. As can be seen in Figure 5.7, the left-hand side of the model is more densely populated than the right-hand side in terms of practices that are assigned to maturity levels 1 to 7, apart from the *Threat Detection & Protection*, *Monetization Strategy* and *Account Management* capabilities. While it may be considered to be desirable to compress the maturity levels as much as possible in order to make the model more compact, the observed distribution of practices is a result of the way the API-m-FAMM is structured due to the dependencies that are present among practices, as is discussed in Section 5.1.1. As a result of juxtaposing practices that have dependencies tied to them, it was concluded that a minimum of 10 maturity levels are needed to structure the API-m-FAMM. As such, the model may not be artificially compressed any further.

Lastly, as a point of discussion, it is important to note that while the API-m-FAMM provides organizations with a structured overview of the topic, API management is often not a straightforward and linear practice to implement. Additionally, as is shown by the case studies that were conducted, even though some practices may be regarded as being more mature, they may not always necessarily be the best fit for an organization. This may often be attributed to a variety of reasons, but can include difficulties with regards to the organization's underlying technical architecture or infrastructure, low priority in comparison to other projects or customer

wishes, or return on investment being too low in relation to the time and resources required to develop and implement the practice.

Furthermore, as some experts mentioned during interviews, the process leading up to the point where APIs are ready to be managed (i.e. design and creation) is often just as, if not more, complex. For example, convincing management and stakeholders of the need for building an API can be a challenge in itself, considering that doing so often requires a substantial investment. Especially since afterwards, resources have to be invested in training employees and evangelizing the API within the organization through internal processes and communication.

## 7.2 Scientific Contribution

Aside from the main practical contributions the API-m-FAMM offers organizations in maturing their API management practices, this work also provides various scientific contributions. These contributions are listed below, and summarized at the end of this section.

- Firstly, this work offers researchers a previously undefined framework that captures the topics and processes API management consists of. During the first stages of this research, mainly as part of the SLR, it soon became clear that relatively little academic literature exists on the topic that goes in-depth with regards to describing the fundamentals of API management. Additionally, little consistency and agreement was observed among researchers as to what elements of API management are considered to be at the core of the topic. Furthermore, most (grey) literature is connected, through one way or another, to commercial API management platform providers such as Google's Apigee and Oracle's WSO2. By decoupling API management processes and topics from such commercial platforms, the API-m-FAMM offers the academic community, as far as the researchers are aware, the first de-commercialized overview of the topic that was developed by using insights from both literature as well as the industry. Due to the fact that this framework is platform independent and constructed in a fully transparent manner, it offers a starting point and stepping stone for future research in the topic of API management.

- Secondly, this work provides researchers that seek to develop a focus area maturity model for a different functional domain with a detailed framework on how to do so. Publications on the development of FAMMs such as that of Jansen (2020) and Spruit and Röling (2014) offer a high-level overview of this process, which adhere to the FAMM meta-model presented by van Steenbergen et al. (2013) and De Bruin et al. (2005)'s methodology for developing maturity models. The steps that this methodology consists of were also followed in this work: conducting a SLR, population, evaluation through expert interviews, and deployment as part of case studies. However, the intermediate incremental versions that were developed throughout the process are not discussed in these publications, as is the case in this work (Chapter 4 & 5). While this was likely a deliberate choice that was made due to constraints in the size and length of academic publications, the fact that the entire development and design process of the API-m-FAMM is visible to the readers of this work contributes to adding to the knowledge base of maturity model development. These insights also contribute towards alleviating concerns mentioned

by Jansen (2020), whom states that: *"Interestingly enough, while there is a rapid increase of publications of new maturity models, there is little literature that particularly discusses the development of maturity models"*. Furthermore, work conducted by Proença and Borbinha (2016) on the state of the art of maturity models has found that one of the main limitations to maturity models is that there is a general *"Lack of information regarding the maturity model development method"*.

- Thirdly, we have incorporated novel approaches in developing the API-m-FAMM. This includes the introduction of classification of the different types of changes that were made to practices and capabilities as a result of grey literature verification (Section 4.2.3): *remove*, *add*, *merge*, *split*, *relocate*, and *rename*. These categories were also utilized to classify the different types of suggestions that were made during expert interviews (Section 5.1.1).

  Furthermore, fundamental decisions that were made towards adjusting the scope and contents of the FAMM following the expert interviews (Section 5.1.1) are elaborated upon, as well as efforts to increase traceability and transparency through the inclusion of detailed figures that show all suggested changes made by experts (Figure 5.3) and the way they were subsequently interpreted (Figure 5.4).

  Moreover, a card sorting technique was used and described, which uses digital tools to rank and assign practices to maturity levels (Section 5.1.1). Additionally, the way these exercises were interpreted and the manner in which practices were ultimately assigned to maturity levels (Section 5.1.1) is elaborated upon.

  Another novel approach includes the utilization of criteria used for DSR artefact evaluation introduced by Prat et al. (2015) to evaluate the API-m-FAMM's usefulness, completeness, ease of use, effectiveness, and operational feasibility. Doing so has shown that using these criteria is an adequate strategy for evaluating FAMMs during expert interviews or through surveys. In this work, the criteria were used to gather feedback and foster discussion with experts as well as among the researchers themselves, in order to subsequently guide further improvements to the API-m-FAMM. While in this case the criteria evaluation was conducted during the first evaluation cycle and as part of the case studies, the criteria could also be used to evaluate a prototype version of a FAMM to gauge interest among practitioners, or as part of the second evaluation cycle.

- Fourthly, this work has shown that De Bruin et al. (2005)'s methodology for maturity model development may be incorporated with DSR, by using it to construct a design science artefact during the *solution design* phase. Furthermore, this study demonstrates which techniques and tools may be utilized during the *test* phase of De Bruin et al. (2005)'s methodology. This includes some of the approaches listed above, such as usage of Prat et al. (2015)'s evaluation criteria, conducting multiple evaluation cycles through expert interviews to evaluate the maturity model, as well as the usage of Nielsen (1995)'s card sorting technique to perform maturity level assignments. As such, this work provides researchers that utilize De Bruin et al. (2005)'s methodology and are involved with maturity model development with suggestions for the usage of novel approaches so that they may incorporate them in the testing phase of their maturity models.

- Fifthly, this work has used a set of categories as part of the case study to collect and determine the status of implementation for each practice contained in the API-m-FAMM. These categories were inspired by the practice implementation categories that are used as part of case studies conducted by Jansen (2020) in evaluating his Focus Area Maturity Model for software ecosystem governance. Initially, the researchers extended the four categories used by Jansen (2020) with a fifth category: *implemented*. This category was introduced to enable data collection regarding the as-is situation of practice implementation in the participating case companies. Having insight into the current status of implementation is paramount towards ensuring that the API-m-FAMM succeeds in achieving its goal, which is to assess the organization's current level of maturity.

  Additionally, in order to highlight the difference between practices that are planned for implementation and those that are not, Jansen (2020)'s *implementable* category was initially renamed to *implementable but not planned* so that it may be used as an alternative to the *implementable and planned* category. However, as is described in Chapter 6.4, during the discussions that were conducted as part of the embedded single-case study it was found that practitioners were unsure as to when a practice may be deemed to be planned for implementation. In order to avoid confusion and ambiguity, these two categories were ultimately merged into the *implementable* category. Furthermore, the decision was made to merge the *not implementable* category with the *not applicable category*, as it became evident from the discussion sessions that these two categories were used interchangeably. All changes that were made to the practice implementation categories are summarized and visualized in Figure 6.3. We consider these changes to be an improvement of the categories that were originally introduced by Jansen (2020), as we have found that the *not implementable* and *implementable and planned* cause ambiguity and confusion among practitioners with regards to aspects such as prioritization and planning of practices, in addition to causing confusion as to what constitutes a practice not being implementable. Researchers involved in research on maturity models and FAMMs may re-use these improved practice implementation categories to collect and present case study data.

- Lastly, the API-m-FAMM was successfully deployed in practice with minimal involvement of the researchers. As is described in Chapter 6, practitioners were provided with the API-m-FAMM, along with a set of instructions and a spreadsheet that was used to denote which practices had been implemented in the case company. To the best of the researchers knowledge, this study is the first among work that has previously been done with regards to the design of FAMMs where practitioners are enabled to self-assess their organization's maturity in a functional domain such as API management. In comparison, Jansen (2020)'s SEG- M2 and Spruit and Röling (2014)'s ISFAM were deployed in practice by gathering input through in-person collaboration between the practitioners and the researchers themselves, as well as desk studies. Moreover, as is discussed in Section 6.3 and shown in Table 6.2, the majority of practitioners' experience with using the API-m-FAMM was positive, considering that its *ease of use*, *usefulness*, and *effectiveness* was ranked with a score of 4 out of 5 on average.

In short, the scientific contributions of this work include a previously undefined

framework of the topic of API management, that is platform and technology independent, de-commercialized, grounded in both literature and practice, as well as constructed in a fully transparent manner. These traits make this framework well-suited for acting as a stepping stone for future research into the topic of API management. Furthermore, considering that all steps involved in the process of developing the API-m-FAMM have been extensively described throughout this work, researchers interested in developing maturity models and FAMMs are also provided with a framework to do so. This includes a classification of the different types of changes that may be made to practices and capabilities, scoping decisions that may be encountered throughout the process, and the usage of the card sorting technique to perform maturity level assignments. Additionally, an improvement of the practice implementation categories originally introduced by Jansen (2020) is proposed. Furthermore, it was shown that the evaluation criteria introduced by Prat et al. (2015) may be used to evaluate FAMMs and can be incorporated into De Bruin et al. (2005)'s methodology for maturity model development, and that FAMMs may be deployed with minimal involvement of the researchers.

## 7.3 Validity

Like all empirical research, this work is vulnerable to threats to validity. Hence, it is of critical importance to analyze and mitigate these threats. Generally speaking, threats to validity may be categorized into four categories: *construct validity*, *internal validity*, *external validity*, and *reliability* (Ampatzoglou, Bibi, Avgeriou, Verbeek, & Chatzigeorgiou, 2019; Runeson & Höst, 2009). In the sections below, several threats to these categories of validity are identified and described, along with the actions that were taken to mitigate them.

### 7.3.1 Construct Validity

Construct validity refers to the degree to which this study actually measures what is is intended to. In the context of this study, this entails assessing an organization's current degree of maturity with regards to API management. This type of validity was mitigated through a number of actions. Firstly, the SLR that was conducted as a means of initially populating the first versions of the API-m-FAMM adhered to several constraints. This includes a search string that was constructed through snowballing in an iterative manner and was peer reviewed by all researchers. Furthermore, multiple databases were searched and strict inclusion and exclusion criteria were adhered to as to mitigate study inclusion and exclusion bias.

Publication bias was further mitigated as a result of the inclusion of grey literature. Additionally, multiple discussion sessions were held among researchers at various stages of the population process in order to mitigate data extraction bias and researcher bias (Ampatzoglou et al., 2019). Moreover, construct validity is mitigated through this work's adherance to DSR guidelines and De Bruin et al. (2005)'s methodology for developing maturity models, as well as providing full transparency in terms of traceability and the design choices that were made throughout the development process. Lastly, robustness of the initial classification that was used by De (2017) was ensured by evaluating this decision through multiple expert interviews. However, while this classification was initially used due to the fact that the SLR found that De (2017)'s work was cited the most often across literature, it should

be noted that the main objects of discussion in this work are API management platforms. While these have been excluded from the scope of the API-m-FAMM, as motivated in Section 5.1.1, more case studies should be conducted at organizations both using or not using API management platforms to investigate whether this decision has negatively impacted construct validity. However, this decision was ran by experts during the second evaluation cycle, during which no criticism or concerns were raised.

### 7.3.2 Internal Validity

Internal validity is concerned with the extent to which there is evidence that the artefact makes a difference in terms of cause and effect in the context of this study. This type of validity is difficult to mitigate in the case of the API-m-FAMM, mainly due to the relatively short duration (8 months) of this study. In order to investigate whether the model is able to achieve its intended goal, which is to assist organizations in assessing and evaluating their degree of maturity in API management in order to improve upon their API management related processes, the researchers suspect that multiple desk studies that last substantial periods of time should be conducted. Similarly, this approach was taken by Jansen (2020) in order to evaluate a FAMM in practice. By combining these desk studies with multiple follow-up interviews, it may be properly investigated whether the usage of the API-m-FAMM by practitioners actually resulted in the implementation of practices that were not previously implemented within the organization, thus fulfilling the artefacts intended goal.

However, verifying the implementation of practices contained in the API-m-FAMM through desk studies is largely infeasible since practices are often placed on long-term road maps that are susceptible to change, reducing the chance of actually observing the implementation of such practices during the desk study. Instead, the effects of the API-m-FAMM were investigated through evaluation with the criteria listed in Table 2.3 during the experts interviews and case studies. Considering the difference and increase in terms of the scores that were assigned to these criteria by practitioners when comparing the first and second version of the API-m-FAMM, in addition to the positive results and feedback received as a result of the case studies, we believe that the model is able to achieve its intended goal. In addition to this long-term goal, short-term benefits may also be attained by using the API-m-FAMM due to practitioners being able to immediately identify practices that are not currently implemented.

### 7.3.3 External Validity

External validity revolves around the degree to which the results of this study may be generalized and applied to other contexts and situations. The API-m-FAMM is mainly targeted towards organizations that expose one or multiple public or partner APIs. However, two experts that are employed at organizations that currently exclusively utilize a set of internal APIs were involved in the first evaluation cycle. Considering that these experts expressed that focus areas such as *Security*, *Observability*, *Performance* and, to a lesser extent, some capabilities belonging to the *Lifecycle* focus area, are relevant and applicable to them, we suspect that the API-m-FAMM is also useful to such organizations.

Organizations of varying sizes and backgrounds were involved in the evaluation cycles and case studies. While some experts that are employed at large organizations

noted that most practices have already been implemented at their organization, they also commented that some practices that are assigned to high maturity levels have not yet been implemented. Furthermore, experts working for small organizations indicated that the API-m-FAMM can be a useful tool for them to use as a road map or checklist to use in discussions with management and stakeholders so that future implementation of practices that are currently not implemented may be discussed and planned. Additionally, due to the decision to exclude practices that are solely tied to the usage of API management platforms, the API-m-FAMM is generalizable to both organizations that do not use such platforms, and those that do not. However, while this was not the case with organizations involved in the evaluation cycles and case studies, more case studies should be conducted with organizations that heavily utilize API management platforms to fully determine whether this exclusion in terms of scoping has negatively impacted the usability of the API-m-FAMM for such such organizations.

### 7.3.4 Reliability

This aspect is concerned with the extent to which the data and analyses that were conducted as part of these study are dependent on the specific researchers. Due to adherence to DSR guidelines, the use of De Bruin et al. (2005)'s methodology, and the SLR, some degree of researcher bias has been mitigated. Furthermore, the protocols that were used for experts interviews and case studies are included and were reviewed through peer reviewing among all involved researchers. Additionally, all design decisions and processes that resulted in the various increments of the API-m-FAMM are extensively documented and described throughout this work, along with separate source documents detailing each major version of the model.

Another threat to reliability is that, considering the maturity level assignments described in Section 5.1.1 were partially done in a pragmatic and subjective matter, they may result in a different outcome if replicated. This is due to the fact that the experts that ranked practices as part of the maturity ranking exercises each have varying backgrounds, experiences, and are employed at organizations with different characteristics. However, this threat was mitigated by using the average of these maturity assessments, as well as analyzing and cross-evaluating focus areas and maturity assignments during the second evaluation cycle. Furthermore, the deployment of the API-m-FAMM as part of the case studies has not produced any criticism among practitioners on the maturity levels that practices are assigned to.

A final potential threat to reliability lies in the fact that organizations that participated in the case studies have used the API-m-FAMM to assess their degree of maturity in API management for themselves. In the context of this study, this was deemed an appropriate approach considering that most of the the participating organizations were also involved in one or both evaluation cycles and were thus familiar with the API-m-FAMM and its contents. As is discussed in Section 6.2.3, the practitioner (whom did not participate in the evaluation cycles) of one case company misinterpreted the case protocol, resulting in the researchers being provided with an incomplete data collection spreadsheet. However, judging from the scores that were assigned to the evaluation criteria, as can be seen in Table 6.2, this misinterpretation has not negatively impacted the practitioners experience with using the API-m-FAMM.

## 7.4 Limitations

Due to the fact that this study concerns a master thesis project, most limitations were caused as a byproduct of time constraints. Firstly, the researchers suspect that full saturation in terms of the amount of capabilities and practices that were uncovered was not able to be reached due to the fact that each of the six focus areas were evaluated three times during the expert interviews. Ideally, focus areas would have been discussed repeatedly until full content saturation had been reached.

Additionally, due to time constraints and the prevalence of the COVID-19 pandemic, desk studies were not a possible method of evaluating the API-m-FAMM's effect in practice over a longer period of time. In comparison, Jansen (2020)'s SEG-M2 FAMM was evaluated through six case studies that comprised 54 interviews and 38 days worth of desk studies in total. However, another example includes Spruit and Röling (2014)'s ISFAM, which was constructed by interviewing 11 domain experts through 13 interviews, and subsequently evaluated through one case study. This is an amount that is comparable to the 11 expert interviews and 6 case studies that were conducted as part of this study.

## 7.5 Opportunities & Future Work

In part with regards to the limitations outlined above, there are some opportunities for future work and improvements that may be identified. Firstly, experts occasionally suggested during interviews that they are of the opinion that a bigger range of practitioners could be reached if the accessibility of the API-m-FAMM as a tool would be improved. One way in which this could be achieved, is by developing a web-app through which practitioners may easily navigate, as well as read focus area, capability, and practice descriptions, and then mark which practices are or are not implemented within their organization. In doing so, the Excel spreadsheet and source document that were used as part of the case studies could be combined into one easily usable tool. Currently, the API-m-FAMM is maintained on `maturitymodels.org`. While this website offers a visual alternative to the source document considering that descriptions are included, it does suffer from some limitations. Most notably, the character length of descriptions is limited, and practitioners are unable to denote whether practices have been implemented. Enabling this would allow practitioners to share the current as-is situation of their organization's API management maturity with management and stakeholders. The opportunity for improved visualization of results extracted from FAMMs was also previously identified by Spruit and Röling (2014). In this work, it is suggested that effective result visualization may be accomplished through spider charts, since most managers are familiar with such a type of representation.

Secondly, as is mentioned in Section 5.1.1, this study was not able to capture the topic of *Governance* in the form of capabilities and practices. While some experts provided some input on practices that could potentially be added, it was decided these would not be of added value to practitioners. Nevertheless, governance is a term and subject in the scope of API management that most experts were familiar with during interviews when asked, in addition to it being relatively prevalent in literature (De, 2017; Jayathilaka, Krintz, & Wolski, 2015; Krintz et al., 2014; Lourenço Marcos & Puccinelli de Oliveira, 2019). Because of this, the researchers suspect that there is an opportunity for future research to perform research to uncover common

threads and patterns in terms of the ways organizations govern their APIs in practice.

Thirdly, the researchers suspect that there is an opportunity to uncover new monetization strategies and unique implementations of monetization models for APIs in the industry that were not yet encountered as part of this study. Considering that these were not observed in practice as part of the conducted case studies, it may be hypothesized that enterprises that are larger than the case companies involved in this study and that have been exposing and working with APIs for a long time, may be utilizing strategies to monetize their APIs.

Furthermore, in the early stages of this research, a goal was set to identify employee roles within organizations that are most commonly involved with each individual practice. As a first step towards this attempt, an overview of roles generally involved in API management was extracted from work by De (2017), Medjaoui et al. (2018), and L. Weir (2019), as shown in Tables 3.2 and 3.3. For future research, it could be investigated and verified whether these roles are actually involved with implementing the practices that are contained in the API-m-FAMM, and whether the naming conventions of these roles match those that are used in practice.

Moreover, future work in this domain could seek to attempt the incorporation of technical implementations and topics that were abstracted away from and excluded from the scope of the API-m-FAMM. This includes the design and creation process of APIs, some of which have been mentioned earlier in this work, such as API first design (Rivero et al., 2013), webhooks (Biehl, 2017), and matters that are of particular concern to developers, which may include elements belonging to development methodologies such as DevOps. The API-m-FAMM could be extended to incorporate these topics, or a separate model may be created.

Another opportunity lies in the potential to customize and adapt the API-m-FAMM depending on certain organizational characteristics and goals. For example, the case studies have shown that certain focus areas are irrelevant for organizations that exclusively utilize internal APIs. Such information could be preemptively collected through a checklist or survey, based on which the contents of the API-m-FAMM may be adapted. Other information that could be used to perform this adaptation may include characteristics such as the size of the organization, whether a third-party API management platform is used, or what type of product or services the organization provides.

The aforementioned opportunity for customization and adaptation of FAMMs can be linked to a general limitation of maturity models, that has been identified in as part of previously conducted research. In his work, Proença and Borbinha (2016) argues that *"maturity assessments can be used to measure the current maturity level of a certain aspect of an organization in a meaningful way, enabling stakeholders to clearly identify strengths and improvement points, and accordingly prioritize what to do in order to reach higher maturity levels"*. However, as a prerequisite to performing such as a maturity assessment, evidence needs to be manually collected to substantiate the maturity level calculation, which makes maturity assessment an an expensive and burdensome activity for organizations. Hence, Proença and Borbinha (2016) argues, future research should focus on developing methods and techniques to automate maturity assessment. The researchers second this observation, as it has been observed that, even though the maturity assessments that were done as part of the case studies were performed with minimal intervention of the researcher, there are opportunities for automation, customization, and adaptation of maturity models and FAMMs to reduce the amount of effort needed to perform maturity assessments and provide tailor-made advice for maturity improvement. Moreover, the researchers

hypothesize that the aforementioned opportunities for automation, customization, and adaptation could be key in creating incentives for practitioners to re-use maturity models and FAMMs throughout a longer period of time. We consider maturity model's low degree of re-usability to be a point of concern due to our findings as part of the case study, which has shown that practitioners are relatively unlikely to revisit the API-m-FAMM to track their progress over a longer period of time.

Lastly, the researchers are of the opinion that useful insights could be gained by conducting research into the differences in terms of advantages and disadvantages that occur with regards to API management for organizations that actively utilize third-party, commercial platforms to manage their APIs when compared to those who do not. This suspicion is supported by the observation that currently, the largest part of available (grey) literature on the topic of API management is either written by authors that are either directly or indirectly affiliated to commercial management platform providers. Examples of such authors include L. Weir (2019) (director at Oracle) and De (2017) (former Apigee consultant). This literature is, more often than not, exclusively focused on the benefits that organization attain as a result of using API management platforms. However, when asked, some experts that were interviewed during the evaluation cycles noted that their organization does not use a management platform and does not wish to do so. For future work, it should be investigated whether significant differences exists in terms of API management maturity between organizations that do use commercial platforms and those that do not.

# Chapter 8

# Conclusion

Throughout this work, the design, population, evaluation, and deployment of a focus area maturity model targeted towards the topic of API management has been described. The goal of this model as well as this work in general was to *improve* the transparency and availability of API management assessment frameworks and tools *by* constructing, evaluating and validating a publicly available, industry and academically grounded framework or tool *that can* be used by organizations that expose their API(s) to third-party developers to assess and evaluate their degree of maturity with regards to API management *in order to* improve upon their API management-related business processes. In order to verify whether this goal has been achieved, the research questions posed in Chapter 1.4 are reflected upon in the following paragraphs.

*MRQ: How can organizations that expose their APIs to third parties evaluate their API management practices?*

To answer this main research question, several sub research questions were posed. The process of answering these individual questions is analyzed, by examining the answers that were formulated throughout this study. After having done so, the main research question is answered in the last paragraph of this chapter.

*SQ1: What type of tool or framework is best suited for organizations wanting to assess their API management related business processes?*

In order to determine which type of tool or framework was best suited for achieving this study's goal, a literature review was first conducted (Chapter 3) to investigate the topic of API management. Additionally, this chapter incorporated a background section which was meant to identify all relevant and existing frameworks, tools, and frameworks that seek to achieve similar goals to those of this study. This process revealed that maturity models and matrices that incorporate maturity degrees, as well as capability comparison matrices, are an often used method for assessing and evaluating organizations' API management related business processes. These findings prompted an investigation into the potential usability of maturity models and capability frameworks, which ultimately resulted in the decision to use a Focus Area Maturity Model (FAMM) to achieve this study's goal. This decision is motivated in Chapter 4. The underlying meta-model of FAMMs proved to be well-suited to capture the various capabilities, activities, and processes API management consists of.

*SQ2: What elements should the API management assessment tool or framework be comprised of?*

As a result of the literature review, it became clear that API management as a whole is a broad topic consisting of a large collection of sub-topics and processes. These were extracted through an extensive SLR that incorporated all relevant literature related to the topic, resulting in a collection of focus areas, capabilities, and practices. As is described in Chapter 4, after having decided on the scope and design of the API-m-FAMM, these were then used to initially populate the model. Through 5 iterations, which were constructed as a result of various categorization and discussion sessions, as well as verification using grey literature, a first version of the model that is grounded in literature was developed. This version consisted of 6 focus areas, 23 capabilities, and 87 practices. The conceptual model representing these focus areas and their corresponding capabilities is shown below in Figure 8.1.

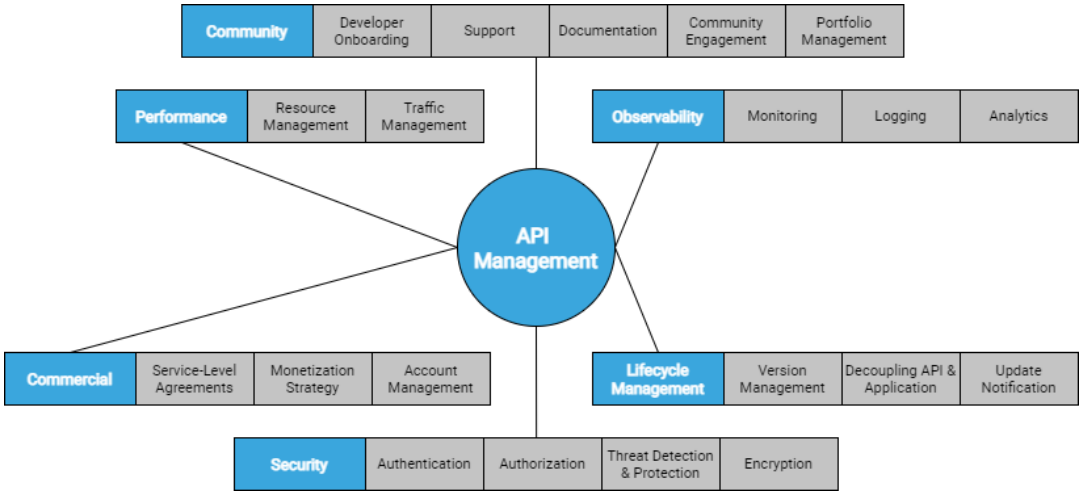



FIGURE 8.1: Conceptual model corresponding to the API-m-FAMM, representing the manner in which the topic of API management is subdivided into 6 focus areas and the 23 capabilities that are assigned to them.

*SQ3: What are suitable criteria, benchmarks, and methods for evaluating the API management assessment tool or framework?*

After having populated and developed the first version of the API-m-FAMM, the model was evaluated. This evaluation was performed using five different evaluation criteria that were adapted from work by Prat et al. (2015): *completeness*, *ease of use*, *effectiveness*, *operational feasibility*, and *usefulness*. As a whole, the evaluation process comprised two evaluation cycles that both utilized expert interviews as a method. These evaluation cycles are described in Chapter 5. During the first of these cycles 11 interviews were conducted with 9 experts, which were mainly focused on evaluating the API-m-FAMM's completeness by discussing all focus areas, capabilities, and practices the first version consisted of. Additionally, all practices were ranked in terms of their perceived maturity, and the model was evaluated on its operational feasibility, ease of use, effectiveness, and usefulness. Many valuable insights were distilled as a result of these interviews. After having analyzed these, the model was modified accordingly through several iterations. This ultimately resulted in the second version of the API-m-FAMM, which is grounded in both literature as well as practice, and consists of 6 focus areas, 20 capabilities, and 81 practices. In order to verify decisions made, this version was then evaluated through 3 expert interviews

as part of the second evaluation cycle.

*SQ4: What is a suitable method for validating the API management assessment tool or framework in practice?*

In order to validate the degree to which the API-m-FAMM succeeds in aiding organizations in evaluating and improving upon their API management related business processes in practice, several case studies were conducted. The case study design consisted of one embedded single-case study and multiple additional case studies, which are described in Chapter 6. By adhering to a case study protocol, the API-m-FAMM was successfully deployed with several case companies to assess their current degree of maturity with regards to API management. Additionally, the API-m-FAMM was evaluated through four criteria: *operational feasibility*, *ease of use*, *usefulness*, and *effectiveness*. The scores assigned that were assigned by practitioners to these criteria indicate that the API-m-FAMM is effective and succesful in achieving its intended goal, which is to aid organizations in assessing their level of maturity in API management and guiding them towards an increased level of maturity. In total, 6 software organizations and products in the Netherlands were included in the case study. Results regarding their respective maturity in API management were benchmarked, discussed, and analyzed, both individually as well as collectively.

As a whole, this research has shown that FAMMs are an appropriate type of maturity model for representing a broad and extensive functional domain such as API management, as well being well-suited for assessing organization's as-is implementation of capabilities and practices. For this research in specific, the API-m-FAMM was proven to be an adequate and efficient tool for aiding organizations in gaining a better understanding of their current implementation of API management practices, and subsequently providing them with guidance towards a higher level of maturity. This statement is supported by the results of the evaluation that was conducted as part of the two evaluation cycles and case studies. It was shown that the utilization of previously established evaluation criteria, that were originally introduced by Prat et al. (2015) and are commonly used for DSR artefact evaluation, are suitable for doing so. Furthermore, it has been shown that FAMMs may be successfully deployed in practice with minimal involvement of the researchers, given that the descriptions of practices, capabilities and focus areas are clearly described, pragmatic, and supplemented with concrete examples, and the instructions for the self-assessment are clearly explained. Additionally, this research helps build understanding among researchers interested in constructing FAMMs as to what concrete steps should be taken, as well as what considerations and dilemmas may be encountered throughout the adherence to De Bruin et al. (2005) methodology for developing maturity models. Furthermore, this work provides scholars and practitioners that are interested in API management with a novel and de-commercialized, as well technology and platform independent overview of the topic that has been grounded in both literature and evaluated in practice. Lastly, this study identifies a set of directions, suggestions, and opportunities for future research. These include suggestions for future directions in API management related research, as well as opportunities for improvement of maturity models.

# References


Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., & Chatzigeorgiou, A. (2019). Identifying, Categorizing and Mitigating Threats to Validity in Software Engineering Secondary Studies. In *Information and software technology* (Vol. 106, pp. 201–230).

Arsanjani, A., & Holley, K. (2006). The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA. In *2006 IEEE International Conference on Services Computing (SCC'06)* (pp. 515–515).

Basole, R. C. (2019). On the Evolution of Service Ecosystems: A Study of the Emerging API Economy. In *Handbook of Service Science* (Vol. 2, pp. 479–495). Springer.

Becker, J., Knackstedt, R., & Pöppelbuß, J. (2009). Developing Maturity Models for IT Management. In *Business & Information Systems Engineering* (Vol. 1, pp. 213–222). Springer.

Biehl, M. (2017). *Webhooks: Events for RESTful APIs* (Vol. 4). API-University Press.

Bonardi, M., Brioschi, M., Fuggetta, A., Verga, E. S., & Zuccalà, M. (2016). Fostering Collaboration Through API Economy: the E015 Digital Ecosystem. In *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice* (pp. 32–38).

Braun, C., Wortmann, F., Hafner, M., & Winter, R. (2005). Method Construction: A Core Approach to Organizational Engineering. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (pp. 1295–1299).

Carmel, E., & Agarwal, R. (2006). The Maturation of Offshore Sourcing of Information Technology Work. In *Information Systems Outsourcing* (pp. 631–650). Springer.

Castillo-Montoya, M. (2016). Preparing for Interview Research: The Interview Protocol Refinement Framework. In *Qualitative report* (Vol. 21, p. 811-831).

Crawford, J. K., et al. (2007). *Project Management Maturity Model*. Taylor & Francis.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. In *IEEE Internet Computing* (Vol. 6, pp. 86–93). IEEE.

De Bruin, T., Rosemann, M., Freeze, R., & Kaulkarni, U. (2005). Understanding the Main Phases of Developing A Maturity Assessment Model. In *Australasian Conference on Information Systems (ACIS)* (pp. 8–19).

Dig, D., & Johnson, R. (2006). How Do APIs Evolve? A Story of Refactoring. In *Journal of software Maintenance and Evolution: Research and Practice* (Vol. 18, pp. 83–107). Wiley Online Library.

Espinha, T., Zaidman, A., & Gross, H.-G. (2014). Web API Growing Pains: Stories from Client Developers and Their Code. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)* (pp. 84–93).

Espinha, T., Zaidman, A., & Gross, H.-G. (2015). Web API Growing Pains: Loosely Coupled Yet Strongly Tied. In *Journal of Systems and Software* (Vol. 100, pp. 27–43). Elsevier.

Etikan, I., Musa, S. A., & Alkassim, R. S. (2016). Comparison of Convenience Sampling and Purposive Sampling. In *American Journal of Theoretical and Applied Statistics* (Vol. 5, pp. 1–4). New York.

Farrell, S. (2009). API Keys to the Kingdom. In *IEEE Internet Computing* (Vol. 13, pp. 91–93). IEEE.

Fielding, R. T. (2000). REST: Architectural Styles and The Design of Network-based Software Architectures. *Doctoral dissertation, University of California*.

Hevner, A., & Chatterjee, S. (2010). Design Science Research in Information Systems. In *Design research in information systems* (pp. 9–22). Springer.

Hevner, A., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. In *MIS Quarterly* (pp. 75–105). JSTOR.

Holley, K., Antoun, S., Arsanjani, A., Brown, W., Cozzi, C., Costas, J., … others (2014). The Power of the API Economy: Stimulate Innovation, Increase Productivity, Develop New Channels, and Reach New Markets. In *IBM Corporate* (Vol. 1, pp. 1–26).

Hüner, K. M., Ofner, M., & Otto, B. (2009). Towards A Maturity Model for Corporate Data Quality Management. In *Proceedings of the 2009 ACM Symposium on Applied Computing* (pp. 231–238).

Iversen, J., Nielsen, P. A., & Norbjerg, J. (1999). Situated Assessment of Problems in Software Development. In *ACM SIGMIS Database: the Database for Advances in Information Systems* (Vol. 30, pp. 66–81). ACM New York, NY, USA.

Jansen, S. (2020). A Focus Area Maturity Model for Software Ecosystem Governance. In *Information and Software Technology* (Vol. 118). Elsevier.

Jones, S., Torres, V., & Arminio, J. (2014). Issues in Analysis and Interpretation. In *Negotiating the Complexities of Qualitative Research in Higher Education: Fundamental Elements and Issues* (pp. 157–173).

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. In *Keele University and Durham University Joint Report.*

March, S. T., & Smith, G. F. (1995). Design and Natural Science Research on Information Technology. In *Decision Support Systems* (Vol. 15, pp. 251–266). North-Holland.

Masse, M. (2011). *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc.

Mathijssen, M., Overeem, M., & Jansen, S. (2020a). Identification of Practices and Capabilities in API Management: A Systematic Literature Review. *arXiv preprint arXiv:2006.10481*.

Mathijssen, M., Overeem, M., & Jansen, S. (2020b). Source Data for the Focus Area Maturity Model for API Management v2. *arXiv preprint arXiv:2007.10611v2*.

Mathijssen, M., Overeem, M., & Jansen, S. (2021a). Source Data for the Focus Area Maturity Model for API Management v3. *arXiv preprint arXiv:2007.10611v3*.

Mathijssen, M., Overeem, M., & Jansen, S. (2021b). Source Data for the Focus Area Maturity Model for API Management v6. *arXiv preprint arXiv:2007.10611v6*.

Maxwell, J. A. (2012). *Qualitative Research Design: An Interactive Approach* (Vol. 41). Sage publications.

Mettler, T. (2011). Maturity Assessment Models: A Design Science Research Approach. In *International Journal of Society Systems Science* (Vol. 3, pp. 81–98). Inderscience Publishers Ltd.

Mettler, T., Rohner, P., & Winter, R. (2010). Towards A Classification of Maturity Models in Information Systems. In *Management of the Interconnected World* (pp. 333–340). Springer.

Myers, B. A., & Stylos, J. (2016). Improving API Usability. In *Communications of the ACM* (Vol. 59, pp. 62–69). ACM.

Mylopoulos, J. (1992). Conceptual Modelling and Telos. In *Conceptual Modelling, Databases, and CASE: An Integrated View of Information System Development* (pp. 49–68). Citeseer.

Nielsen, J. (1995). Card Sorting to Discover the Users' Model of the Information Space. In *Nielsen Norman Group.*

Okoli, C. (2015). A Guide to Conducting a Standalone Systematic Literature Review. In *Communications of the Association for Information Systems* (Vol. 37, p. 43).

Patton, M. Q. (2014). *Qualitative Research & Evaluation Methods: Integrating Theory and Practice*. Sage publications.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability Maturity Model, Version 1.1. In *IEEE Software* (Vol. 10, pp. 18–27). IEEE.

Polya, G. (2004). *How To Solve It: A New Aspect of Mathematical Method* (Vol. 85). Princeton University Press.

Pöppelbuß, J., & Röglinger, M. (2011). What Makes A Useful Maturity Model? A Framework of General Design Principles for Maturity Models and its Demonstration in Business Process Management.

Prat, N., Comyn-Wattiau, I., & Akoka, J. (2015). A Taxonomy of Evaluation Methods for Information Systems Artifacts. In *Journal of Management Information Systems* (Vol. 32, pp. 229–267). Taylor & Francis.

Proença, D., & Borbinha, J. (2016). Maturity models for Information Systems: A State of the Art. In *Procedia Computer Science* (Vol. 100, pp. 1042–1049). Elsevier.

Rivero, J. M., Heil, S., Grigera, J., Gaedke, M., & Rossi, G. (2013). MockAPI: An Agile Approach Supporting API-first Web Application Development. In *International Conference on Web Engineering* (pp. 7–21).

Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers. In *IEEE Software* (Vol. 26, pp. 27–34). IEEE.

Rubin, H., & Rubin, I. (2011). *Qualitative Interviewing: The Art of Hearing Data*. Sage publications.

Runeson, P., & Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. In *Empirical Software Engineering* (Vol. 14, p. 131). Springer.

Salvadori, I., & Siqueira, F. (2015). A Maturity Model for Semantic Restful Web APIs. In *2015 IEEE International Conference on Web Services* (pp. 703–710).

Shrestha, A., Cater-Steel, A., & Toleman, M. (2014). How To Communicate Evaluation Work in Design Science Research? An Exemplar Case Study.

Spruit, M., & Röling, M. (2014). ISFAM: The Information Security Focus Area Maturity Model.

Stylos, J. (2009). *Making APIs More Usable With Improved API Designs, Documentation and Tools* (Unpublished doctoral dissertation). Carnegie Mellon University.

van de Weerd, I., & Brinkkemper, S. (2009). Meta-modeling for Situational Analysis and Design Methods. In *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 35–54). IGI Global.

van Steenbergen, M., Bos, R., Brinkkemper, S., van De Weerd, I., & Bekkers, W. (2010). The Design of Focus Area Maturity Models. In *International Conference on Design Science Research in Information Systems* (pp. 317–332).

van Steenbergen, M., Bos, R., Brinkkemper, S., van de Weerd, I., & Bekkers, W. (2013). Improving IS Functions Step by Step: the Use of Focus Area Maturity Models. In *Scandinavian J. Inf. Systems* (Vol. 25, p. 2).

Venable, J., Pries-Heje, J., & Baskerville, R. (2012). A Comprehensive Framework

for Evaluation in Design Science Research. In *International Conference on Design Science Research in Information Systems* (pp. 423–438).

Vukovic, M., Laredo, J., Muthusamy, V., Slominski, A., Vaculin, R., Tan, W., ... others (2016). Riding and Thriving on the API Hype Cycle. In *Communications of the ACM* (Vol. 59, pp. 35–37). ACM New York, NY, USA.

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer.

Wilde, E., & Pautasso, C. (2011). *REST: From Research to Practice*. Springer Science & Business Media.

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and A Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 1–10).

Xavier, L., Brito, A., Hora, A., & Valente, M. T. (2017). Historical and Impact Analysis of API Breaking Changes: A Large-scale Study. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 138–147).

Yin, R. K., et al. (2003). Design and Methods. In *Case Study Research* (Vol. 3).

# Grey Literature


Anji. (2018). *Web API Versioning Techniques.* Retrieved 2020-07-09, from http://learninghubspot.com/index.php/2018/01/19/webapi-versioning/

Apigee. (2020). *Managing API product bundles.* Retrieved 2020-07-06, from https://docs.apigee.com/api-platform/monetization/api-product-bundles

Auth0. (2021a). *OAuth Scopes.* Retrieved 2021-01-03, from https://auth0.com/docs/sso

Auth0. (2021b). *Single Sign-On.* Retrieved 2021-01-03, from https://auth0.com/docs/sso

Averdunk, I., & Moen, H. K. (2020). *Implement Health Check APIs for Microservices.* Retrieved 2020-07-10, from https://www.ibm.com/garage/method/practices/manage/health-check-apis/

Barracuda. (2020). *What Is Failover?* Retrieved 2020-12-16, from https://www.barracuda.com/glossary/failover

Bhojwani, R. (2018). *Backward Compatibility Check for REST APIs.* Retrieved 2020-07-10, from https://dzone.com/articles/backward-compatibility-check-for-rest-apis

Budzynski, M. (2016). *How to Monetize APIs with Azure API Management.* Retrieved 2020-07-10, from https://azure.microsoft.com/en-us/blog/how-to-monetize-apis-with-azure-api-management/

CA Technologies. (2019). *The API Management Playbook: Understanding Solutions for API Management.* Retrieved 2020-09-21, from https://docs.broadcom.com/docs/the-api-management-playbook

Castellani, S., & Dorairajan, A. (2020). *What Are the Different Types of APIs?* Retrieved 2020-08-28, from https://apifriends.com/api-creation/different-types-apis/

Coleman Parkes Research. (2017). *APIs: Building A Connected Business in the App Economy.* Retrieved 2020-08-11, from https://docs.broadcom.com/doc/apis-building-a-connected-business-in-the-app-economy

Crowe, C. (2018). *What is An API Consumer?* Retrieved 2020-08-06, from https://community.apigee.com/questions/43660/what-is-an-api-consumer.html

Devoteam. (2016). *API Management at Liberty Global Inc.* Retrieved 2020-09-21, from https://nl.devoteam.com/en/our-case-studies/api-management-liberty-global-inc

Dropbox. (2021). *Data Transport Limit.* Retrieved 2021-01-03, from https://www.dropbox.com/developers/reference/data-transport-limit

Endjin. (2017). *API Maturity Matrix.* Retrieved 2020-09-21, from https://endjin.com/blog/2017/07/kickstart-your-api-proposition-with-the-api-maturity-matrix

Forrester Research. (2020). *The Forrester Wave™: API Management Solutions, Q3 2020.* Retrieved 2020-08-11, from https://wso2.com/resources/analyst-reports/the-forrester-wave-api-management-solutions-q3-2020/

Gartner. (2019). *A Guidance Framework for Evaluating API Management Solutions.* Retrieved 2020-09-21, from https://www.gartner.com/en/documents/3956412/a-guidance-framework-for-evaluating-api-management-solut

GraphQL. (2020). *GraphQL Best Practices.* Retrieved 2020-07-09, from https://graphql.org/learn/best-practices/

Guerrero, H., & Ramos, V. (2016). *API Versioning Methods: A Brief Reference.* Retrieved 2020-07-09, from https://developers.redhat.com/blog/2016/06/13/api-versioning-methods-a-brief-reference/

Haddad, C. (2015). *Comparison Evaluation Matrix.* Retrieved 2020-09-18, from https://wso2.com/whitepapers/comparison-evaluation-matrix/

Hosting Manual. (2020). *SLA & Uptime Calculator.* Retrieved 2020-07-13, from https://www.hostingmanual.net/uptime-calculator/

Kubernetes. (2021). *Tools for Monitoring Resources.* Retrieved 2021-01-03, from https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/

Lees, C., Mallick, A., Paar, T., Fontenay, E., & Pimakova, K. (2019). *APIs: The Digital Glue - How Banks Can Thrive in an API Economy.* Retrieved 2020-09-16, from https://www.accenture.com/_acnmedia/PDF-100/Accenture-How-Banks-Can-Thrive-API-Economy.pdf

Madhvani, A. (2018). *API Versioning: An Overview.* Retrieved 2020-07-09, from https://icapps.com/blog/api-versioning-overview

Moiz, A. (2020). *Service Level Agreement (SLA) Monitoring and Reporting.* Retrieved 2020-07-10, from https://www.extnoc.com/blog/service-level-agreement-monitoring-reporting/

Mueller, J. P. (2020). *What Is an API Sandbox?* Retrieved 2020-07-06, from https://smartbear.com/learn/api-design/what-is-an-api-sandbox/

Mulesoft. (2020). *Types of APIs and How to Determine Which to Build.* Retrieved 2020-08-28, from https://www.mulesoft.com/resources/api/types-of-apis

Onelogin. (2020). *How Does Single Sign-on Work?* Retrieved 2020-12-16, from https://www.onelogin.com/learn/how-single-sign-on-works

OpenID. (2021). *Welcome to OpenID Connect.* Retrieved 2021-01-03, from https://openid.net/connect/

Oracle. (2020). *WS-Security Authentication.* Retrieved 2020-07-09, from https://www.oracle.com/topics/technologies/ws-audit-authentication.html

OWASP. (2021a). *A1:2017-Injection.* Retrieved 2021-01-03, from https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

OWASP. (2021b). *A3:2017-Sensitive Data Exposure.* Retrieved 2021-01-03, from https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

Peter, B. (2017). *API Lifecycle, Versioning, and Deprecation.* Retrieved 2020-07-10, from https://zapier.com/engineering/api-geriatrics/

Ploesser, K. (2019). *Best Practices for Versioning REST and GraphQL APIs.* Retrieved 2020-07-09, from https://www.moesif.com/blog/technical/api-design/Best-Practices-for-Versioning-REST-and-GraphQL-APIs/

Postman. (2019). *Postman State of the API Report.* Retrieved 2020-10-12, from https://www.postman.com/resources/infographics/api-survey-2019/

RapidAPI. (2020). *API Versioning – What is API Versioning?* Retrieved 2020-07-09, from https://rapidapi.com/blog/api-glossary/api-versioning/

Redhat. (2020). *What Is API Monetization?* Retrieved 2020-07-10, from https://www.redhat.com/en/topics/api/what-is-api-monetization

Reynold, D. (2020). *Cybersecurity Breach Protocol: Balancing Legal and Communication*

*Risks.* Retrieved 2020-12-16, from https://boardmember.com/cybersecurity-breach-protocol-balancing-legal-and-communication-risks/

Salesforce. (2020). *AppExchange Security Review.* Retrieved 2020-12-16, from https://developer.salesforce.com/docs/atlas.en-us.packagingGuide.meta/packagingGuide/security_review_overview.htm

Sandoval, K. (2018a). *3 Common Methods of API Authentication Explained.* Retrieved 2020-12-16, from https://nordicapis.com/3-common-methods-api-authentication-explained/

Sandoval, K. (2018b). *Methods To Communicate API Change Effectively.* Retrieved 2020-07-10, from https://nordicapis.com/methods-to-communicate-api-change-effectively/

Sandoval, K. (2018c). *Methods To Communicate API Change Effectively.* Retrieved 2020-07-06, from https://nordicapis.com/methods-to-communicate-api-change-effectively/

Santos, W. (2019). *APIs Show Faster Growth Rate in 2019 Than Previous Years.* Retrieved 2020-08-11, from https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/research/2019/07/17

Soliya. (2020). *Security and Breach Protocols.* Retrieved 2020-12-16, from https://www.soliya.net/security-and-breach-protocols

Swagger. (2018). *Welcome To Swagger Spec Compatibility's Documentation!* Retrieved 2020-07-10, from https://swagger-spec-compatibility.readthedocs.io/en/latest/

TU Delft. (2018). *Template Informed Consent Form.* Retrieved 2020-08-26, from https://www.tudelft.nl/over-tu-delft/strategie/integriteitsbeleid/human-research-ethics/template-informed-consent-form/

Tung, T. (2014). *Accenture API Management Suite API Maturity Model.* Retrieved 2020-09-15, from https://expydoc.com/doc/4730985/download-pdf

Tyk. (2020). *Enforced Timeouts.* Retrieved 2020-07-10, from https://tyk.io/docs/planning-for-production/ensure-high-availability/enforced-timeouts/

Uptrends. (2021). *7 Alerting Optimizations You Should Use in Your Website and API Monitoring.* Retrieved 2021-01-03, from https://blog.uptrends.com/technology/7-alerting-optimizations-you-should-use-in-your-website-and-api-monitoring/

Utrecht University. (2020). *Informed Consent for Data Sharing.* Retrieved 2020-08-26, from https://www.uu.nl/en/research/research-data-management/guides/informed-consent-for-data-sharing

Wikipedia. (2019). *WS-Security.* Retrieved 2020-07-09, from https://en.wikipedia.org/wiki/WS-Security

Wikipedia. (2020). *Zero Trust Networks.* Retrieved 2020-12-16, from https://en.wikipedia.org/wiki/Zero_Trust_Networks

Wikipedia. (2021). *XACML.* Retrieved 2021-01-03, from https://en.wikipedia.org/wiki/XACML

WSO2. (2015). *API Management Platform Technical Evaluation Framework.* Retrieved 2020-09-18, from https://wso2.com/whitepapers/api-management-platform-technical-evaluation-framework/

WSO2. (2020). *Monitoring HTTP Access Logs.* Retrieved 2020-07-10, from https://apim.docs.wso2.com/en/3.0.0/administer/product-administration/monitoring/logging/monitoring-http-access-logs/

# Citations Belonging with the SLR


Akbulut, A., & Perros, H. G. (2019). Software Versioning with Microservices through the API Gateway Design Pattern. In *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 289–292).

Andrey Kolychev, K. Z. (2019). Studying Open Banking Platforms with Open Source Code.

Biehl, M. (2015). *API Architecture: the Big Picture for Building APIs*. CreateSpace.

Bui, D. H. (2018). *Design and Evaluation of a Collaborative Approach for API Lifecycle Management* (Unpublished master's thesis).

Ciavotta, M., Alge, M., Menato, S., Rovere, D., & Pedrazzoli, P. (2017). A Microservice-based Middleware for the Digital Factory. In *Procedia Manufacturing* (Vol. 11, pp. 931–938). Elsevier.

Coste, R., & Miclea, L. (2019). API Testing for Payment Service Directive2 and Open Banking. In *International Journal of Modeling and Optimization* (Vol. 9).

De, B. (2017). *API Management*. Springer.

Familiar, B. (2015). IoT and Microservices. In *Microservices, IoT, and Azure* (pp. 133–163). Springer.

Fremantle, P., Kopeckỳ, J., & Aziz, B. (2015). Web API Management Meets the Internet of Things. In *European Semantic Web Conference* (pp. 367–375).

Gadge, S., & Kotwani, V. (n.d.). Microservice Architecture: API Gateway Considerations. In *GlobalLogic Organisations,*.

Gámez Díaz, A., Fernández Montes, P., & Ruiz Cortés, A. (2015). Towards SLA-driven API Gateways. In *XI Jornadas De Ciencia E IngenierÍA De Servicios.*

Hofman, W., & Rajagopal, M. (2014). A Technical Framework for Data Sharing. In *Journal of Theoretical and Applied Electronic Commerce Research* (Vol. 9, pp. 45–58). Universidad de Talca.

Hohenstein, U., Zillner, S., & Biesdorf, A. (2018). Architectural Considerations for a Data Access Marketplace Based upon API Management. In *International Conference on Data Management Technologies and Applications* (pp. 91–115).

Indrasiri, K., & Siriwardena, P. (2018). Developing Services. In *Microservices for the enterprise* (pp. 89–123). Springer.

Jacobson, D., Woods, D., & Brail, G. (2011). *APIs: A Strategy Guide*. O'Reilly Media.

Jayathilaka, H., Krintz, C., & Wolski, R. (2015). EAGER: Deployment-time API Governance for Modern PaaS Clouds. In *2015 IEEE International Conference on Cloud Engineering* (pp. 275–278).

Krintz, C., Jayathilaka, H., Dimopoulos, S., Pucher, A., Wolski, R., & Bultan, T. (2014). Cloud Platform Support for API Governance. In *2014 IEEE International Conference on Cloud Engineering* (pp. 615–618).

Lourenço Marcos, E., & Puccinelli de Oliveira, R. (2019). A Framework for Guidance of API Governance: A Design Science Approach.

Matsumoto, O., Kawai, K., & Takeda, T. (2017). Fujitsu Cloud Service K5 PaaS Digitalizes Enterprise Systems. In *Fujitsu Scientific & Technical Journal* (Vol. 53, pp. 17–24).

Medjaoui, M., Wilde, E., Mitra, R., & Amundsen, M. (2018). *Continuous API Management: Making the Right Decisions in an Evolving Landscape*. O'Reilly Media.

Montesi, F., & Weber, J. (2016). Circuit Breakers, Discovery, and API Gateways in Microservices. *arXiv preprint arXiv:1609.05830*.

Nakamura, N. (2017). Fujitsu's Leading Platform for Digital Business. In *Fujitsu Scientific & Technical Journal* (Vol. 53, pp. 3–10).

Patni, S. (2017). *Pro RESTful APIs*. Springer.

Preibisch, S. (2018). API Gateways. In *API Development* (pp. 125–142). Springer.

Raivio, Y., Luukkainen, S., & Seppala, S. (2011). Towards Open Telco-Business Models of API Management Providers. In *2011 44th Hawaii International Conference on System Sciences* (pp. 1–11).

Siné, M., Haezebrouckl, T.-P., & Emonet, E. (2015). API-AGRO: An Open Data and Open API Platform to Promote Interoperability Standards for Farm Services and Ag Web Applications. In *Journal of Agricultural Informatics* (Vol. 6, pp. 56–64). Hungarian Association of Agricultural Informatics.

Šnuderl, M. (2018). *Rate Limiting in API Management* (Unpublished doctoral dissertation). University of Ljubljana, Faculty of Computer and Information Science.

Thielens, J. (2013). Why APIs Are Central to a BYOD Security Strategy. In *Network Security* (Vol. 8, pp. 5–6). Elsevier.

Vijayakumar, T. (2018). *Practical API Architecture and Development with Azure and AWS: Design and Implementation of APIs for the Cloud*. Apress.

Weir, L. (2019). *Enterprise API Management: Design and Deliver Valuable Business APIs*. Packt Publishing Ltd.

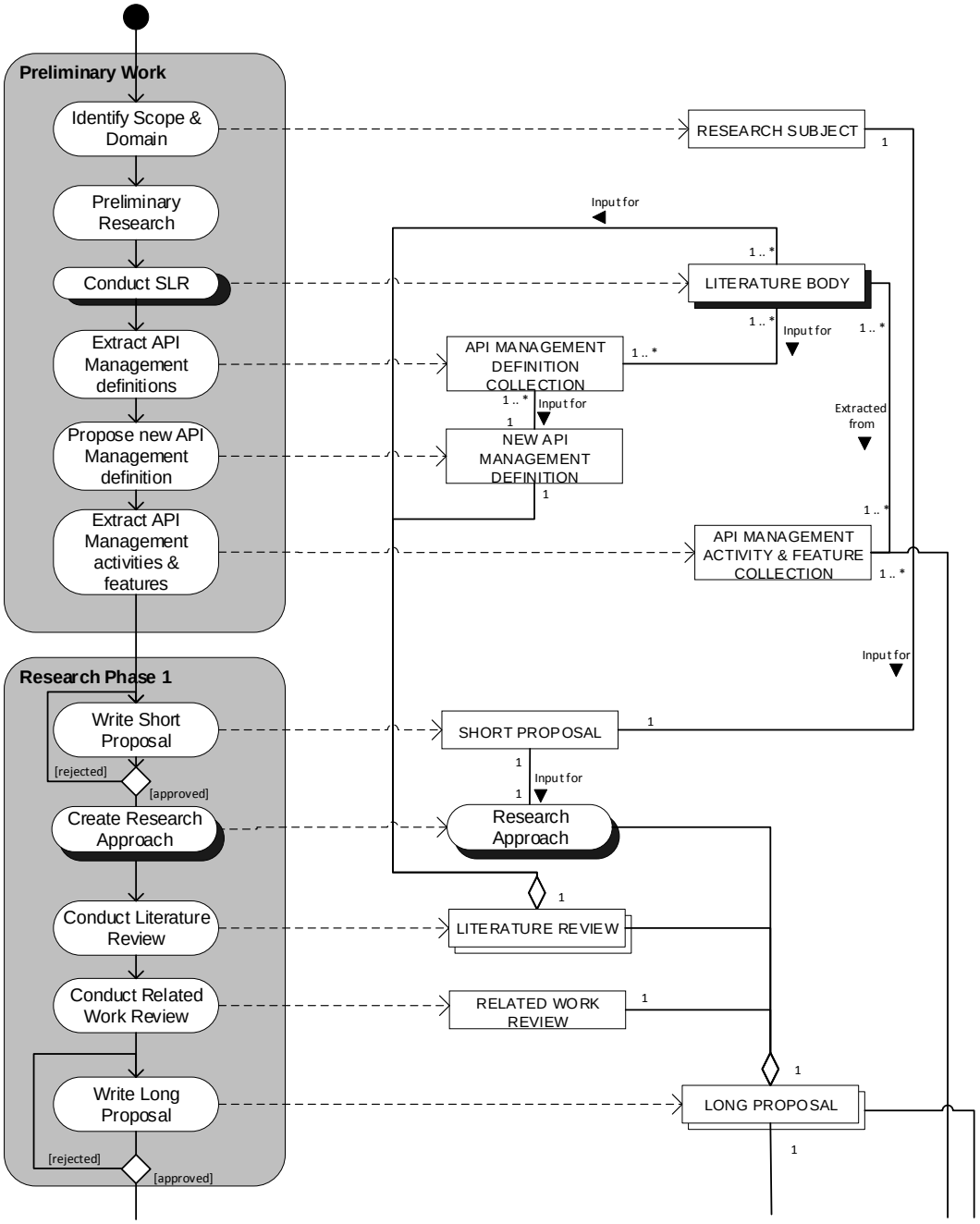
Weir, L. A., Bell, A., Carrasco, R., & Viveros, A. (2015). *Oracle API Management 12c Implementation*. Packt Publishing Ltd.

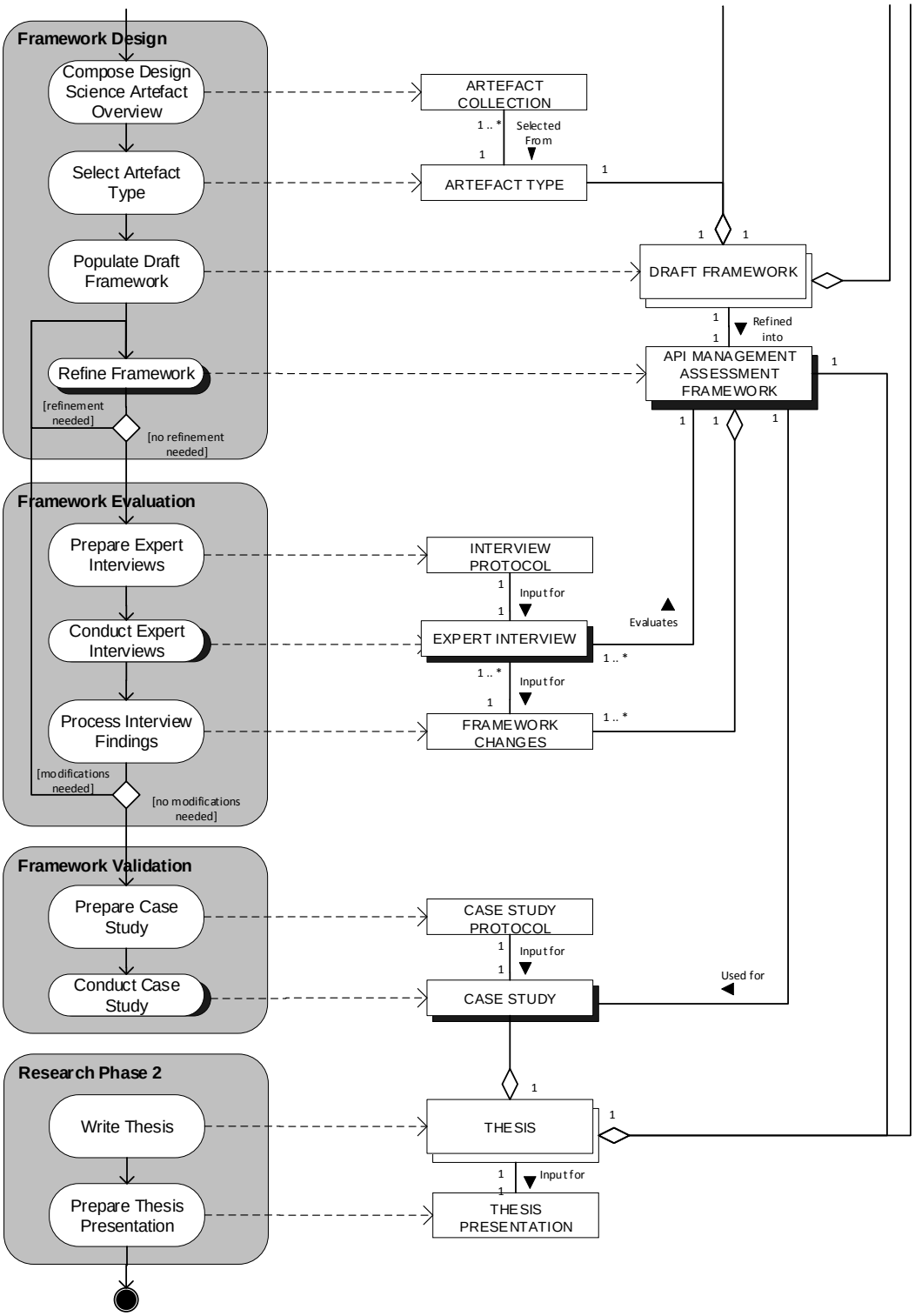
Xu, R., Jin, W., & Kim, D. (2019). Microservice Security Agent Based On API Gateway in Edge Computing. In *Sensors* (Vol. 19, p. 4905). MDPI AG.

Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018). Management of API Gateway Based on Micro-service Architecture. In *Journal of Physics: Conference Series* (Vol. 1087, p. 32). IOP Publishing.

# Appendix A

# Process Deliverable Diagram




Framework Design
Compose Design Science Artefact Overview
Select Artefact Type
Populate Draft Framework
Refine Framework
[refinement needed]
[no refinement needed]
Framework Evaluation
Prepare Expert Interviews
Conduct Expert Interviews
Process Interview Findings
[modifications needed]
[no modifications needed]
Framework Validation
Prepare Case Study
Conduct Case Study
Research Phase 2
Write Thesis
Prepare Thesis Presentation
ARTEFACT COLLECTION
Selected From
ARTEFACT TYPE
DRAFT FRAMEWORK
Refined into
API MANAGEMENT ASSESSMENT FRAMEWORK
INTERVIEW PROTOCOL
Input for
EXPERT INTERVIEW
Evaluates
FRAMEWORK CHANGES
CASE STUDY PROTOCOL
CASE STUDY
Used for
THESIS
THESIS PRESENTATION

# Appendix B

# Interview Protocol

## B.1 Interview Protocol Checklist

| Interview Protocol Aspects | Yes | No | Feedback for improvement |
|---|---|---|---|
| ***Interview Protocol Structure*** | | | |
| Beginning questions are factual in nature | ☐ | ☐ | |
| Key questions are majority of the questions and are placed between beginning and ending questions | ☐ | ☐ | |
| Questions at the end of interview protocol are reflective and provide participant an opportunity to share closing comments | ☐ | ☐ | |
| A brief script throughout the interview protocol provides smooth transitions between topic areas | ☐ | ☐ | |
| Interviewer closes with expressed gratitude and any intents to stay connected or follow up | ☐ | ☐ | |
| Overall, interview is organized to promote conversational flow | ☐ | ☐ | |
| ***Writing of Interview Questions & Statements*** | | | |
| Questions/statements are free from spelling error(s) | ☐ | ☐ | |
| Only one question is asked at a time | ☐ | ☐ | |
| Questions are mostly open ended | ☐ | ☐ | |
| Questions are written in a non-judgmental manner | ☐ | ☐ | |
| ***Length of Interview Protocol*** | | | |
| All questions are needed | ☐ | ☐ | |
| Questions/statements are concise | ☐ | ☐ | |
| ***Comprehension*** | | | |
| Questions/statements are devoid of academic language | ☐ | ☐ | |
| Questions/statements are easy to understand | ☐ | ☐ | |

## B.2 Preliminary Survey

Thank you for showing interest in potentially taking part in an interview to evaluate the first version of our API management assessment framework. In order to determine whether you fit the desired characteristics for participating in our research, we would like to ask you a few quick questions. First, please answer the questions below regarding your current role in your organization and the amount of years you have been working with APIs or in the field of API management. If you have any questions, please contact one of the researchers:

m.mathijssen@students.uu.nl
slinger.jansen@uu.nl
michiel.overeem@afas.nl

1. What is your e-mail address?
2. What is your current role within your organization?
3. Can you briefly describe how often and in what way you work with APIs? PS: feel free to answer this question in Dutch!
4. How many years worth of experience do you have with either consuming, developing, integrating, providing, versioning, monitoring or managing APIs? (0 - 10+)

**API Management Knowledge**

Thank you for answering the two previous questions. In order to further determine whether you fit the desired characteristics for participating in our research, we would like to measure your knowledge on various API management-related concepts. Before answering the following questions, we would like to ask you to please briefly read through the descriptions below, corresponding to the six main topics the API management assessment framework consists of. After having read the description of a topic, please answer the question attached.

*Lifecycle*

Generally speaking, an API undergoes several stages over the course of its lifetime; creation, publication, realization, maintenance and retirement. In order to control and guide the API through these stages, the organization must be able to perform a variety of activities. At its core, the organization must be able to set the various stages in motion by being able to create API endpoints and proxies, publish or import APIs and deprecate it when needed. In order to maintain the API, the organization must decide on a versioning strategy, notification channels and methods in case of updates, perform interface translation and have design- and run-time policies in place. In doing so, the organization is able to manage and maintain the versions the API goes through as it evolves over time.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Security*

Due to their programmatic nature and their accessibility over the public cloud, APIs are prone to various kinds of attacks. Hence, the organization should undertake various measures to prevent this from happening. For example, one of many available authentication and authorization protocols should be implemented, prevention for attacks such as DoS or SQL script injection attacks should be in place and sensitive data should be encrypted or masked.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Performance*

The overall performance of a client app is dependent on the performance of the underlying APIs powering the app. Hence, the importance of performance for APIs increases greatly. In order to ensure performance and stability of their APIs, organizations must be able to perform various activities. For example, being able to perform caching improves an API's performance through reduced latency and network traffic. Additionally, using rate limiting and throttling mechanisms to manage traffic and using connection pooling and load balancing to route traffic more effectively also improves the API's performance.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Monitoring*

As an organization, it is necessary to have insight into the API program to make the right investments and decisions during its maintenance. Through various monitoring techniques, the organization is able to collect metrics which can shed light on the API's health and performance, which in turn, improve the decision making process on how to enhance the business value by either changing the API or by enriching it. Additionally, by being able to log API access, consumption and performance, input may be gathered for analysis, business value or monetization reports. These may be used to strengthen communication with consumers and stakeholders or check for any potential service-level agreement violations.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Community Engagement*

As an organization exposing APIs for external consumers and developers to consume, it is often desirable to foster, engage and support the community that exists around the API. For example, this entails offering developers the ability register on the API and offering them access to test environments, code samples and documentation. Additionally, the organization may support developers in their usage of the API by offering them support through a variety of communication channels and allowing them to communicate with the organization or among another through a community forum or developer portal. Furthermore, it is desirable for developers to be able to freely browse through the API offering, review operational status updates regarding the API, create support tickets in the event of an error and to share knowledge, views and opinions with other developers.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Community Engagement*

As an organization exposing APIs for external consumers and developers to consume, it is often desirable to foster, engage and support the community that exists around the API. For example, this entails offering developers the ability register on the API and offering them access to test environments, code samples and documentation. Additionally, the organization may support developers in their usage of the API by offering them support through a variety of communication channels and allowing them to communicate with the organization or among another through a community forum or developer portal. Furthermore, it is desirable for developers to be able to freely browse through the API offering, review operational status updates regarding the API, create support tickets in the event of an error and to share knowledge, views and opinions with other developers.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

*Commercial*

Oftentimes, exposing and consuming APIs has a commercial aspect tied to it. For API consumers and providers, this is often embodied by legal business contracts for the use of the APIs which they are bound to. These business contracts called service-level agreements govern the service levels and other aspects of API delivery and consumption. Another commercial aspect of API management is that of monetization. Considering APIs provide value to the consuming party, organizations often opt to monetize the services and APIs and build a business model for them. Utilizing the right monetization model for APIs enables organizations to reap the benefits of their investment in their APIs.

- How experienced or knowledgeable are you on the topic described above? (1 / not at all - 5 / very much)

Thank you for your participation in this preliminary survey. We will contact you as soon as possible to either invite you to participate in an interview or to inform you as to why you do not fit our desired characteristics, and to thank you for your interest and time.

## B.3 Interview Information Sheet

Universiteit Utrecht

### Interview Information Sheet

Research Project Title: *Designing an API Management Assessment Framework (conducted by Max Mathijssen, supervised by Dr. Slinger Jansen (Utrecht University) and Michiel Overeem (AFAS Software)*
Contact: m.mathijssen@students.uu.nl, slinger.jansen@uu.nl
Interview Participant:
Version Date: 18-09-2020

#### Purpose and Context of Study

You are being invited to participate in a research study titled *Designing an API Management Assessment Framework*. This study is being conducted by Max Mathijssen, as part of his master's thesis project in the Business Informatics programme at Utrecht University, under supervision of Dr. Slinger Jansen (Utrecht University) and Michiel Overeem (AFAS Software).

The purpose of this research study is to improve the transparency and availability of existing API management assessment frameworks and tools by constructing, evaluating, and validating a publicly available as well as industry and academically grounded framework. This framework can be used by organizations that expose their API(s) to third-party developers to assess and evaluate their degree of maturity with regards to API management, in order to improve upon their API management-related business processes.

Currently, the first version of this framework consists of six main topics that comprise several capabilities and activities. By using these as a checklist, organizations can check whether these activities have been implemented. Depending on the amount of implemented activities, organizations can determine their degree of maturity regarding API management.

This expert interview consists of 3 parts:

1. First, you will be asked to briefly describe your background and current role at your organization.
2. Secondly, you will be introduced to the current version of the API management assessment framework. Next, you will be shown all the capabilities and activities corresponding to one of the main topics you are knowledgeable on. After having done so, you will be asked a series of questions regarding the *completeness* of these capabilities and activities.
3. Finally, you will be asked a series of general closing questions.

#### Withdrawal from Study

In case you wish to withdraw from this study at any time for whatever reason, you can contact us via e-mail. Doing so is also possible after having participated in the interview. Should you wish to withdraw, any information provided as part of the interview will be deleted and not included in this research's output. The withdrawal request should occur within 30 days after the interview has taken place.

If you have read the above information and wish to participate in an interview, please continue to the next page of this document to review and sign the informed consent upon agreeing with the statements listed.

## B.4 Interview Informed Consent Form

Universiteit Utrecht

### Interview Consent Form

Research Project Title: *Designing an API Management Assessment Framework*

| **Please tick the appropriate boxes** | **Yes** | **No** |
|---|---|---|
| **1.** I have been able to ask questions about the study and my questions have been answered to my satisfaction. | O | O |
| **2.** I consent voluntarily to be a participant in this study and understand that I can refuse to answer questions and I can withdraw from the study at any time, without having to give a reason. | O | O |
| **3.** I understand that information I provide will be used for the evaluation of the API Management Assessment Framework as part of Max Mathijssen's Master Thesis. | O | O |
| **4.** I understand that taking part in the study involves taking part in an audio-recorded interview, which will be transcribed as text and destroyed after transcription. | O | O |
| **5.** I Understand that the principal researcher will keep a link that identifies you to my coded information, but this link will be kept secure and available only to the principal researcher or selected members of the research team. Any information that can identify me will remain confidential. | O | O |
| **6.** I have been given the explicit guarantee that my personal data will be processed in full compliance with the Utrecht University Personal Data Processing Policy. | O | O |
| **7.** I give permission for the information extracted from the interviews that I provided to be used for the creation of Max Mathijssen's Master Thesis to be stored in the Utrecht University thesis archive, so that it can be used for future learning and research. | O | O |
| **8.** I understand that I am free to contact any of the people involved in this research to seek further clarification and information. | O | O |
| **9.** I have carefully read and fully understood the points and statements in this form. All my questions were answered to my satisfaction, and I voluntarily agree to participate in this interview. | O | O |
| **10.** I have obtained a copy of this consent form, which is co-signed by the interviewer. | O | O |

**Signatures**

| | | |
|---|---|---|
| Name of participant | _____ Signature | _____ Date |
| Researcher name | _____ Signature | _____ Date |

Study contact details for further information:
[*Max Mathijssen, m.mathijssen@students.uu.nl. Slinger Jansen, slinger.jansen@uu.nl*]

## B.5 Interview Protocol Mapping Matrix

| Mapping Interview Questions to Research Questions | Interviewee Background | MRQ: How can organizations that expose their API(s) to third parties evaluate their API management practices? | | | Evaluation Criteria | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Interview Questions | | SQ1: What type of tool or framework is best suited for organizations wanting to assess their API management related business processes? | SQ2: What elements should the API management assessment tool or framework be comprised of? | SQ3: What are suitable criteria and benchmarks for validating the API management assessment tool or framework? | Completeness: To what degree does the API management assessment framework or tool contain all the (best) practices, capabilities and features API Management is comprised of? | Effectiveness: To what degree does the API management assessment tool or framework succeed in aiding organizations to improve upon their API management-related business processes? | Ease of Use: To what degree is it free of effort for an organization to self-assess and evaluate their degree of maturity with regards to API management, using the API management assessment tool or | Operational Feasibility: To what degree are experts expecting organizations to utilize the API management assessment tool or framework in practice? | Usefulness: To what degree does the API management assessment tool or framework provide organizations with valuable and useful insights regarding their API |
| **Introduction Phase (Survey)** | | | | | | | | | |
| What is your current role within your organization? | X | | | | | | | | |
| Can you briefly describe how often and in what way you work with APIs? | X | | | | | | | | |
| How many years worth of experience do you have with either consuming, developing, integrating, providing, versioning, monitoring or | X | | | | | | | | |
| **Content Evaluation Phase (for each Capability and corresponding Practices out of selected Focus Areas)** | | | | | | | | | |
| *Focus Area* : Do you think any capabilities are missing? | | | X | | X | | | | |
| *Capability* : Do you recognize this capability as it is described? | | | X | | X | | | | |
| *Capability* : Do you think this capability is listed under the correct Focus Area? | | | X | | X | | | | |
| *Capability* : Do you think any practices are missing? | | | X | | X | | | | |
| *Practice* : Do you recognize this practice as it is described? | | | X | | X | | | | |
| *Practice* : Do you think this practice is listed under the correct Capability & Focus Area? | | | X | | X | | | | |
| *Practice*: Can you think of anything (e.g. other/missing practices, conditions) this practice might depend on? | | | X | | X | | | | |
| *Practice*: Out of all practices in this capability, which practice do you think is the least mature? | | | X | | X | | | | |
| *Practice*: Out of all practices in this capability, which practice do you think is the most mature? | | | X | | X | | | | |

| Closing Phase (General Open Questions) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Do you think any capabilities or practices are missing from the Focus Areas you have been shown? | | X | | X | | | | | | |
| What name and description would you assign to the maturity levels? | X | | | X | | | | | | |
| Do you think it would be better to assess an organization per capability or per focus area? | X | | | | | | | X | | |
| Do you think using this model could help organizations improve their API management related business processes? *(1 - Not at all, 5- Absolutely)* | X | | | | X | | | | X | |
| How easy do you think it would be for an organization to use this model to self-assess their API management related business processes? *(1 - Very difficult, 5 - Very easy)* | X | | | | | X | | | | |
| Do you think you would ever use this model to self-assess your organization's API management related business processes? *(1 - Never, 5 - Absolutely)* | X | | | | | | | X | | |
| Do you think this model is easy to understand? *(1 - Very difficult, 5- Very easy)* | X | | | | | X | | | | |
| Do you have any questions, suggestions for improvement or remarks? | | | | | | | | | | |

## B.6 Interview Protocol



**Interview Protocol**

**Script prior to interview**

I would like to thank you once again for your willingness to participate in this interview. As was mentioned in the Interview Information Sheet you have previously been sent, the purpose of this research study is to improve the transparency and availability of existing API management assessment frameworks and tools by constructing, evaluating, and validating a publicly available as well as industry and academically grounded framework. This framework can be used by organizations that expose their API(s) to third-party developers to assess and evaluate their degree of maturity with regards to API management, in order to improve upon their API management-related business processes.

Currently, the first version of this framework consists of six main topics that comprise several capabilities and activities. By using these as a checklist, organizations can check whether these activities have been implemented. Depending on the amount of implemented activities, organizations can determine their degree of maturity regarding API management.

Our interview will last approximately one and a half hours and will cover multiple questions on the topics you have indicated to be knowledgeable on as part of the previously sent survey. Do you have any questions you would like to ask me before we start?

- *If yes*: answer questions.
- *If no*: proceed with interview.

**Background information interviewee**

I would like to start by verifying the answers you have given regarding your background, as part of the previously sent survey.

1. What is your current role within your organization?
2. Can you briefly describe how often and in what way you work with APIs?
3. How many years' worth of experience do you have with either consuming, developing, integrating, providing, versioning, monitoring or managing APIs?

**Content Evaluation Phase**

Next, I would like to discuss some of the main topics the model consists of.
Begin with showing picture of first Focus Area as a whole, and briefly describe it. Next, show picture of first capability corresponding to this Focus Area. Then, describe the capability.

1. Do you recognize this capability as it is described?
2. Do you think this capability is listed under the correct Focus Area?

Repeat this for all capabilities, until all capabilities have been shown and described.

3. Do you think any capabilities are missing?

Start with first capability and show a picture of it, along with the respective practices it is composed of. Describe the first (least mature) practice.

4. Do you recognize this practice as it is described?
5. Do you think this practice is listed under the correct Capability & Focus Area?
6. Can you think of anything (e.g. other/missing practices, conditions) this practice might depend on?
7. Can you rank the practices on their degree of maturity? (Drag & Drop)

Repeat this for every practice the capability is composed of. Then, move on to next capability and repeat. When last capability is discussed, move on to next Focus Area. Start at question 1 and repeat.

**Closing Phase (General Open Questions)**

Thank you for your answers so far. Next, I would like to ask you a few closing questions.

1. Do you think any capabilities or practices are missing from the Focus Areas you have been shown?
2. What name and description would you assign to the maturity levels?
3. Do you think it would be better to assess an organization per capability or per focus area?
4. Do you think using this model could help organizations improve their API management related business processes? (1 - Not at all, 5 - Absolutely)
5. How easy do you think it would be for an organization to use this model to self-assess their API management related business processes? (1 - Very difficult, 5 - Very easy)
6. Do you think you would ever use this model to self-assess your organization's API management related business processes? (1 – Never, 5 - Absolutely)
7. Do you think this model is easy to understand? (1 - Very difficult, 5 - Very easy)
8. Do you have any questions, suggestions for improvement or remarks?
9. Ask any follow-up questions on interesting responses given throughout the interview.

**Closing Remarks**

Thank you very much for participating in this interview and helping us to improve our framework. If you are interested in our research; would you be willing to help us improve the contents of our framework by briefly reading through the descriptions of the capabilities and practices? If so, we could send you a document on which you can give some feedback, after which you can send it back to us. We would very much appreciate this, but of course, this is not mandatory in any shape or form. Thank you again for your participation.

# Appendix C

# Experts & Interviews Summaries

## C.1 API Evangelist

*API Evangelist* is currently working independently as a product manager for an organization that supplies climate systems. In this role, he is focused on the internal community, which entails determining how APIs should be configured, maintained and used by teams. Furthermore, he has worked as an API evangelist for the community of a large organization that provides Enterprise Resource Planning (ERP) software. Activities as part of this previous role involved advising the organization on how to configure APIs, and requesting the development for APIs that the community asked for. Additionally, *API Evangelist* is currently involved in advisory projects with another organization, where he also acts as an API evangelist. These projects involve evaluating and encouraging banks to incorporate the usage of APIs in their business processes. Because of his experience with the commercial and community engagement aspects of API management, the *Community Engagement* and *Commercial* focus areas were selected to be discussed. Notable input gathered as a result of this interview included suggestions for addition of practices and capabilities related to community involvement and management.

## C.2 CEO A

$CEO_A$ is one of the founders of a small organization that provides customers with a project management tool, which is used by companies to allocate employees and resources to projects of their choosing. Previously, $CEO_A$ was responsible for product acceptance, implementation, and support. Currently, he is involved with sales. Considering that the organization's software is often integrated with other systems, the organization provides customers with a public API through which anyone with the proper rights is able to access data. Additionally, the organization has an internal API on which their software is built. Because of his experience with providing support to consumers of the organization's public API, the *Community Engagement* focus area was selected to be discussed. However, $CEO_A$ indicated that because of the fact that his organization only serves a small amount of customers and hence does not have a substantial community surrounding its API, and only provides one public API, he does not have sufficient knowledge with regards to capabilities such as community management and portfolio management. However, interesting suggestions for the addition of practices belonging to the documentation capability were made.

## C.3 CEO B

$CEO_B$ acts as the managing director of Trancon, which is a small organization that he is also the partial owner of, but has been the CTO for the largest part of his time with the organization. He is responsible for operations, HR, support, development, and architecture. Trancon supplies warehouse management software for logistics, which is used to integrate with ERP systems. Furthermore, the organization develops internal APIs for their own devices to use. Additionally, the organization also uses various APIs internally to handle business processes such as providing support. Two interviews were conducted with $CEO_B$, due to the fact that he possessed knowledge on a variety of focus areas due to his wide range of responsibilities within his organization. These interviews yielded a large amount of interesting insights, including many suggestions for addition of practices regarding version management.

## C.4 Engineer

*Engineer* works as a software engineer for Uber, which is a large-scale multinational organization that provides customers with a transportation application. He is a part of a development team that is mainly responsible for optimizing the performance and scalability of the public API that is provided by the organization, as well as developing new functionalities. Because of the activities *Engineer* is involved in, the *Performance* and *Monitoring* focus areas were selected for evaluation. This interview resulted in new suggestions for the addition of a variety of practices related to scaling, as well as monitoring and load balancing techniques.

## C.5 IT Consultant

*IT Consultant* works for Devoteam, and is a member of the IT strategy consultancy team which specializes in API management. Devoteam advises a variety of companies that differ in terms of scale and size, and is partnered with several large API management platform providers such as Apigee, Akana, and Mulesoft. *IT Consultant* has advised multiple companies, which for example, involved him acting as the lead architect in implementing an enterprise API management platform for a large telecom provider. Additionally, he has been responsible for maintaining, managing, and upgrading API management platforms, as well as having conducted many training projects on behalf of his organization. During these training sessions and workshops, *IT Consultant* educates practitioners on the relevance, benefits, and importance of various concepts related to API management. Because of his experience in consultancy and implementing various API management platforms from different vendors, *IT Consultant* is able to provide insight that aids in discerning what aspects of API management are platform or tool specific, and those that are generic elements of the topic. Hence, two interviews were conducted with *IT Consultant*, which were focused on the evaluation of the *Commercial*, *Community Engagement*, *Security*, and *Lifecycle* focus areas. These interviews yielded many interesting suggestions and comments, such as the addition of several practices related to design-time and run-time governance, error handling, and authentication.

## C.6 Product Manager

*Product Manager* is currently working as a commercial product manager for Exact, an organization that develops and provides Enterprise Resource Planning (ERP) software. His responsibilities include maintaining the product's vision, and managing the product's lifecycle and strategy. The product is composed of a partner API, including the ecosystem surrounding it, and the organization's mobile strategy. *Product Manager* is involved with ensuring that customers use the API as frequently as possible, as well as communicating customer wishes to the strategic and executive teams, and driving new strategic partnerships that involve API integrations. Furthermore, he is involved with the security architecture of the product. Because of his activities, the *Security* and *Commercial* focus areas were selected for evaluation. However, the latter focus area was not discussed due to a lack of time. Notable findings from this interview included suggestions for addition of capabilities and practices related to transparency, authentication, and threat protection.

## C.7 Lead Engineer A

*Lead Engineer*$_A$ is a lead engineer for a large consultancy organization, as part of the IT risk advisory team. This team is responsible for developing software products that are sold to customers. His role involves all technical aspects of developing these products, as well as leading the development team. *Lead Engineer*$_A$'s team uses internal APIs, third-party service integrations to access data from service providers, and also is in the process of starting to expose (partner) APIs to customers. Because of this transition his organization is making towards exposing partner and public APIs, *Lead Engineer*$_A$ noted that he is currently involved with many practices that are assigned to the *Performance* and *Monitoring* focus areas. Hence, these focus focus areas were selected for evaluation as part of this interview.

## C.8 Lead Engineer B

*Lead Engineer*$_B$ is the head of engineering at an organization that partners with leisure companies, whom make use of venue management systems, to extract information regarding current availability from their software and then place new bookings. To do so, the organization utilizes several APIs internally, and provides its integration partners with an 'inventory API' so that they can communicate their availability statuses to the organization. *Lead Engineer*$_B$ is responsible for software development, as well as development and maintenance of APIs. Because of his technical experience, and his experience in managing partner relationships and integrations, the *Commercial* and *Performance* focus areas were selected for evaluation as part of this interview. Notable findings included the addition of several capabilities and practices related to the commercial aspect of API management, as well as SLAs.

## C.9 Lead Engineer C

*Lead Engineer*$_C$ works for a food delivery organization, where he is the lead engineer of two product teams. These teams develop multiple internal APIs, that are used to integrate with other services and clients. Additionally, he is responsible for ensuring that these APIs are up to date, versioned when needed, and monitored sufficiently.

The organization is well-versed and strict in enforcing backwards compatibility and avoiding breaking changes. Considering his experience with versioning and lifecycle management, the *Lifecycle* focus area was selected to be evaluated. The resulting interview with *Lead Engineer*$_C$ yielded in the validation of several previously unconfirmed practices relating lifecycle control, as well as additional information on several practices.

# Appendix D

# Descriptions of Focus Areas, Capabilities & Practices

## D.1 Focus Areas & Capabilities

1. **Lifecycle Management**: Generally speaking, an API undergoes several stages over the course of its lifetime; creation, publication, realization, maintenance and retirement (Medjaoui et al., 2018). In order to control and guide the API through these stages, the organization must be able to perform a variety of activities. In order to maintain the API, the organization must decide on a versioning strategy, notification channels and methods in case of updates, as well as decouple their API from their application. In doing so, the organization is able to manage and maintain the versions the API goes through as it evolves over time.

    1.1 *Version Management*: APIs evolve over time with newer business requirements. In order to cope with this, the organization should have a versioning strategy in place, such as managing multiple versions of an API to support existing consumers, or by avoiding breaking changes as part of an evolutionary strategy. Additionally, the organization should be able to deprecate and retire older versions of their API smoothly. With proper notice and period, deprecated APIs should be retired and removed so as to avoid any maintenance overheads (De, 2017). In order to guide this process, the organization may also have a deprecation protocol in place.

    1.2 *Decoupling API & Application*: When an organization creates an API to expose its data and services, it needs to ensure that the API interface is intuitive enough for developers to easily use (De, 2017). However, the interface for the API will most likely be different from that of the back-end services that it exposes. Therefore, the organization should be able to transform the API interface to a form that the back end can understand.

    1.3 *Update Notification*: Changes made to an API may adversely affect its consumers. Hence, consumers must be notified of any planned updates of the API (De, 2017). The organization should have the ability to inform developers using the API of any changes by distributing change logs, using a communication channel such as email, the developer portal, or preemptively through the use warning headers or a versioning roadmap.

2. **Security**: APIs provide access to valuable and protected data and assets (De, 2017). Therefore, security for APIs is necessary to protect the underlying assets

from unauthenticated and unauthorized access. Due to the programmatic nature of APIs and their accessibility over the public cloud, they are also prone to various kinds of attacks. Hence, the organization should undertake various measures to prevent this from happening. For example, one of many available authentication and authorization protocols should be implemented, prevention for attacks such as DoS or SQL script injection attacks should be in place and sensitive data should be encrypted or masked.

2.1 *Authentication*: Authentication is the process of uniquely determining and validating the identity of a client (De, 2017). In order to achieve this, the organization may implement an authentication mechanism such as API keys or protocols such as WSS or OpenID Connect, or the Single Sign-on method.

2.2 *Authorization*: Authorization controls the level of access that is provided to an app making an API call and controls which API resources and methods that can invoke (De, 2017). The organization may implement authorization through access control or an industry-standardized authorization protocol such as OAuth 2.0.

2.3 *Threat Detection & Protection*: The likelihood of bad actors making attacks using malicious content is high, in addition to common threats such as DoS attacks. Content-based attacks can be in the form of malformed XML or JSON, malicious scripts, or SQL within the payload (De, 2017). Therefore, the organization should be able to detect malformed request formats or malicious content within the payload and then protect against such attacks.

2.4 *Encryption*: Oftentimes, message payloads sent in API calls contain sensitive information that can be the target for man-in-the-middle attacks (De, 2017). Therefore, the organization should secure all communication between the client app and the API service through using techniques such as TLS encryption by default. Furthermore, it is desirable for the organization to prevent exposure of sensitive data by making utilizing methods such as masking or hashing.

3. **Performance**: APIs are no longer exclusively seen as mechanisms for integration but have become mainstream for the delivery of data and services to end users through various digital channels (De, 2017). This increases the demand on APIs to perform well under loads. The overall performance of a client app is dependent on the performance of the underlying APIs powering the app. Hence, the importance of performance for APIs increases greatly. In order to ensure performance and stability of their APIs, organizations must be able to perform various activities. For example, enabling consumers to implement caching improves an API's performance through reduced latency and network traffic. Additionally, using rate limiting and throttling mechanisms to manage traffic and using load balancing to route traffic more effectively also improves the API's performance.

3.1 *Resource Management*: In order to improve the performance of their API(s), it is important for an organization to effectively manage the available resources. This may be accomplished through the use of mechanisms such as load balancing, scaling, or by having a failover policies in place.

3.2 *Traffic Management*: Another aspect of improving API performance is effectively managing incoming traffic. In order to do so, the organization may choose to implement mechanisms such as caching, rate limiting or throttling, or by prioritizing traffic based on customer characteristics.

4. **Observability**: As an organization, it is necessary to have insight into the API program to make the right investments and decisions during its maintenance. Through various monitoring techniques, the organization is able to collect metrics which can shed light on the API's health, performance and resource usage. In turn, these metrics may be aggregated and analyzed to improve the decision making process on how to enhance the business value by either changing the API or by enriching it (De, 2017). Additionally, by being able to log API access, consumption and performance, input may be gathered for analysis, business value or monetization reports. These may be used to strengthen communication with consumers and stakeholders or check for any potential service-level agreement violations.

4.1 *Monitoring*: As an organization, it is important to be able to collect and monitor metrics and variables concerning the exposed API. For example, information regarding the health and performance of the API, as well as resources used by the API should be monitored so that it may be used as input for activities such as generating analysis reports and broadcasting the API's operational status.

4.2 *Logging*: In monitoring their API(s), it is helpful for the organization to be able to perform logging of consumer behavior and activities. This may include logging of API access, usage and reviewing historical information.

4.3 *Analytics*: As an organization, it is important to be able to analyze the metrics and variables that are collected through monitoring. For example, information regarding the health and performance of the API may be utilized to decide which features should be added to the API. Additionally, it is desirable for the organization to be able to extract custom variables from within the message payload for advanced analytics reporting.

5. **Community**: As an organization exposing APIs for external consumers and developers to consume, it is often desirable to foster, engage and support the community that exists around the API. For example, this entails offering developers the ability register on the API and offering them access to test environments, code samples and documentation. Additionally, the organization may support developers in their usage of the API by offering them support through a variety of communication channels and allowing them to communicate with the organization or among another through a community forum or developer portal. Furthermore, it is desirable for developers to be able to freely browse through the API offering, review operational status updates regarding the API, create support tickets in the event of an error and to share knowledge, views

and opinions with other developers.

5.1 *Developer Onboarding*: To start consuming APIs, developers must first register with the organization that is providing them. The sign up process should be simple and easy, possibly by supporting developers with resources such as (automatically generated) SDKs and testing tools such as an API console or sandbox environment.

5.2 *Support*: In order to strengthen the community around the API, the organization should support developers whom are consuming it. This may be accomplished by establishing an appropriate communication channel, adequately managing issues and handling errors, should they present themselves.

5.3 *Documentation*: API documentation can help speed up the adoption, understanding and effectiveness of APIs (De, 2017). Hence, the organization must provide consumers of their API(s) with reference documentation. Additionally, they may be supplied with start-up documentation, code samples and FAQs to further accelerate understanding of the API.

5.4 *Community Management*: Oftentimes, app developers wish to know the views of other developers in the community. They may want to collaborate and share their API usage learnings and experiences with one another (De, 2017). In order to facilitate these wishes, the organization may choose to provide developers with a community forum or developer portal.

5.5 *Portfolio Management*: As an API providing organization, a platform to publicize and document APIs is needed. Hence, a discoverable catalog of APIs through which potential consumers are able to browse may be provided.

6. **Commercial**: Organizations have been consuming third-party APIs to simplify and expand business partnership. APIs provide faster integration and an improved partner/customer experience, enabling organizations to grow rapidly (De, 2017). Oftentimes, exposing and consuming APIs has a commercial aspect tied to it. For API consumers and providers, this is often embodied by legal business contracts for the use of the APIs which they are bound to. These business contracts called service-level agreements govern the service levels and other aspects of API delivery and consumption. Another commercial aspect of API management is that of monetization. Considering APIs provide value to the consuming party, organizations often opt to monetize the services and APIs and build a business model for them (De, 2017). Utilizing the right monetization model for APIs enables organizations to reap the benefits of their investment in their APIs.

6.1 *Service-Level Agreements*: A service-level agreement (SLA) defines the API's non-functional requirements, serving as a contract between the organization and consumers of their API. As such, the organization should ensure that the consumer of their API agrees with the SLA's contents. These may include matters such as terms and conditions for API usage, consumption quotas, uptime guarantees and maintenance or downtime information.

6.2 *Monetization Strategy*: APIs securely expose digital assets and services that are of value to consumers. Hence, the organization may wish to adopt a monetization strategy to enable monetization of the exposed services and APIs by constructing a business model around them. This may be accomplished through a monetization model which can be based on consumer characteristics such as their type of subscription, access tier or the amount of resources used.

6.3 *Account Management*: It is desirable to effectively manage accounts in order to foster a qualitative relationship with customers, stakeholders and the organization's management. This may be achieved by reporting on the API's business value internally through the use of business value reports, as well as externally by providing consumers of the API with subscription reports and training them in using the API as efficiently as possible.

## D.2 Practices

Lifecycle Management — Version Management

| | |
|---|---|
| **Practice Code**: 1.1.2 | **Practice Name**: Implement Evolutionary API Strategy |
| ***Description:*** | The organization utilizes an evolutionary strategy to continuously version their API over time. Using this strategy, the organization evolves a single API by avoiding the introduction of breaking changes. Optionally, this may be accomplished by adhering to the GraphQL specification (GraphQL, 2020). |
| ***Implemented when:*** | • The organization maintains one version of their API.<br>• The organization utilizes an evolutionary API versioning strategy. |
| Literature: | (Madhvani, 2018; Ploesser, 2019) |

| | |
|---|---|
| **Practice Code**: 1.1.5 | **Practice Name**: Implement Multiple API Versioning Strategy |
| ***Description:*** | The organization has a versioning strategy in place which entails the process of versioning from one API to a newer version. In order to do so, the organization must be able to maintain multiple versions of (one of) their API(s) for a period of time. Possible strategies include URI/URL Versioning (possibly in combination with adherence to the Semantic Versioning specification), Query Parameter versioning, (Custom) Header versioning, Accept Header versioning or Content Negotiation. |
| ***Implemented when:*** | • The organization utilizes one of the following versioning strategies: URI/URL Versioning, Query Parameter versioning, (Custom) Header versioning, Accept Header versioning or Content Negotiation. |
| Literature: | (De (2017); (Anji, 2018; Guerrero & Ramos, 2016; RapidAPI, 2020) |

| | |
|---|---|
| **Practice Code**: 1.1.6 | **Practice Name**: Implement API Deprecation Protocol |
| ***Description:*** | The organization has a protocol in place that details what steps should be taken when deprecating one of their APIs. This includes determining the amount of developers currently consuming the API through the use of monitoring, and then setting a threshold that details the amount of developers that should have migrated to the new version of the API before commencing with deprecation of the old version. Furthermore, developers, including their contact information, should be identified so that they may be notified of the deprecation through their preferred communication channel. This notification should be accompanied by a migration period and deprecation date, so that consumers have a clear target to migrate their apps over to the new API version. Additionally, referrals to to documentation and the new endpoint should be included. Furthermore, the protocol should detail what course of action should be taken to roll back to a previously deployed version of an API in the event of an incorrect deployment of the API. |
| ***Implemented when:*** | • The organization has implemented the 'Distribute Versioning Notification Through Channel(s)' (1.3.3) and 'Log Activity' (4.2.3) practices.<br>• The organization has a deprecation protocol in place. |
| Literature: | (Peter, 2017) |

| | |
|---|---|
| **Practice Code**: 1.1.7 | **Practice Name**: Check Backwards Compatibility |
| ***Description:*** | The organization has an approach in place with which it is able to detect breaking changes when versioning their API(s). Approaches include using a unit test suite, plugging an automated contract test suite into the CI/CD pipeline or by using the *swagger-spec-compatibility* library to detect differences between two Swagger / OpenAPI specifications (Swagger, 2018). |
| ***Implemented when:*** | • The organization has implemented the 'Implement Evolutionary API Versioning Strategy' (1.1.2) practice.<br>• The organization has a backwards compatibility checking approach in place. |
| Literature: | (Bhojwani, 2018) |

Lifecycle Management — Decoupling API & Application

| | |
|---|---|
| **Practice Code**: 1.2.1 | **Practice Name**: Decouple API & Software Versioning |
| ***Description:*** | The organization has decoupled the version of their API(s) from its software implementation. The API version should never be tied to the software version of the back-end data/service. A new API version should be created only if there is a change in the contract of the API that impacts the consumer. |
| ***Implemented when:*** | • The organization has decoupled the version of their API(s) from its software implementation. |
| Literature: | (De, 2017) |

| | |
|---|---|
| **Practice Code**: 1.2.4 | **Practice Name**: Decouple Internal & External Data Model |
| ***Description:*** | The organization has decoupled the data model that is used internally, as well as externally. Doing so is considered to be beneficial, since an application might use a normalized relation data model internally. While this data model is less suitable to expose through a public API, this separation of concerns allows the organization to evolve the relational data model at a different speed than the API. |
| ***Implemented when:*** | The organization has decoupled the data model that is used internally, as well as externally. |
| Literature: | None. |

| | |
|---|---|
| **Practice Code**: 1.2.5 | **Practice Name**: Decouple Internal & External Data Format |
| ***Description:*** | The organization has decoupled the data format that is used internally, as well as externally. Doing so is considered to be beneficial, since an application might use a data format such as XML internally, while using a data format such as JSON for the API(s). This separation of concerns grants the organization greater flexibility in designing and developing their APIs. |
| ***Implemented when:*** | • The organization has decoupled the data format that is used internally, as well as externally. |
| Literature: | None. |

| | |
|---|---|
| **Practice Code**: 1.2.6 | **Practice Name**: Decouple Internal & External Transport Protocol |
| ***Description:*** | The organization has decoupled the transport protocol that is used internally, as well as externally. Considering that an application might internally use a protocol that is less commonly used in modern APIs such as SOAP or JDBC internally, which may be less suitable for public APIs, the organization may opt to use a different protocol for their API(s). This separation of concerns grants the These protocols are less commonly used in modern APIs, or are less suitable for public APIs, and the organization can decide to use a different protocol for the APIs. This separation of concerns grants the organization greater flexibility in designing and developing their APIs. |
| ***Implemented when:*** | • The organization has decoupled the transport protocol that is used internally, as well as externally. |
| Literature: | None. |

Lifecycle Management — Update Notification

| **Practice Code**: 1.3.2 | **Practice Name**: Distribute Changelogs |
|---|---|
| ***Description:*** The organization uses (automated) email services to distribute changelogs describing the versioning of their API(s) to consumers. Ideally, the organization offers consumers the ability to opt-in or opt-out of this service. | |
| ***Implemented when:*** • The organization uses (automated) email services to distribute changelogs describing the versioning of their API(s) to consumers. | |
| Literature: (Sandoval, 2018b) | |

| **Practice Code**: 1.3.3 | **Practice Name**: Distribute Versioning Notification Through Channel(s) |
|---|---|
| ***Description:*** The organization has the ability to distribute versioning notifications among consumers of their API(s) through established communication channels. Possible channels include email, social media, and announcements within the developer portal or reference documentation. Ideally, the organization offers consumers of their API(s) the option to select the communication channel they prefer receiving versioning notifications through. | |
| ***Implemented when:*** • The organization has implemented the 'Establish Communication Channel' (5.2.1) and 'Distribute Changelogs' (1.3.2) practices. • The organization has the ability to distribute versioning notifications among consumers of their API(s) through established communication channels. | |
| Literature: (De (2017); Sandoval (2018b) | |

| **Practice Code**: 1.3.5 | **Practice Name**: Extend API with Versioning Information |
|---|---|
| ***Description:*** The organization has the ability to extend their API specification to incorporate warning headers into responses in run-time. By doing so, consumers of the API are notified of its impending deprecation, and possibly requested to change their implementation. | |
| ***Implemented when:*** • The organization has implemented the 'Provide API Status Page', 'Reference Documentation Standard Used' or 'Developer Portal' practice. • The organization has the ability to introduce warning headers. | |
| Literature: (De, 2017) | |

| **Practice Code**: 1.3.9 | **Practice Name**: Announce Versioning Roadmap |
|---|---|
| ***Description:*** The organization has announced a roadmap that details the planned dates on which the current (old) version of their API will be versioned to a new version, in order to notify consumers ahead of time. This may be done through email, social media, announcements within the developer portal or reference documentation. | |
| ***Implemented when:*** • The organization has implemented the 'Distribute Versioning Notification Through Channel(s)' (1.3.3) practice. • The organization has announced a versioning roadmap. | |
| Literature: (De, 2017) | |

Security — Authentication

**Practice Code**: 2.1.1 | **Practice Name**: Implement Basic Authentication

***Description:*** The organization has the ability to implement basic authentication in order to authenticate consumers of their API(s). This may be accomplished through the use of HTTP Basic Authentication, with which the consumer is required to provide a username and password to authenticate, or by issuing API keys to consumers of the API. An app is identified by its name and a unique UUID known as the API key, often serving as an identity for the app making a call to the API.

***Implemented when:***

- The organization has implemented HTTP Basic Authentication, or is able to issue API keys.

Literature: (Biehl (2015); De (2017); Zhao, Jing, and Jiang (2018); (Sandoval, 2018a)

**Practice Code**: 2.1.4 | **Practice Name**: Implement Authentication Protocol

***Description:*** The organization has implemented an authentication protocol or method in order to authenticate consumers of their API(s). In order to apply security For SOAP APIs, the usage of a WS Security (WSS) protocol (Wikipedia, 2019) may be opted for. This protocol specifies how integrity and confidentiality can be enforced on messages and allows the communication of various security token formats, such as Security Assertion Markup Language (SAML), X.509 and User ID/Password credentials. Consumers of REST APIs may be authenticated by using methods and protocols such as Client Certificate authentication, SAML authentication, or OpenID Connect (OpenID, 2021). OpenID Connect 1.0 is an authentication protocol that builds on top of OAuth 2.0 specs to add an identity layer. It extends the authorization framework provided by OAuth 2.0 to implement authentication.

***Implemented when:***

- The organization has implemented a WSS authentication protocol, or methods and protocols such as Client Certificate authentication, SAML authentication, or OpenID Connect.

Literature: (De (2017); (Oracle, 2020; Wikipedia, 2019)

**Practice Code**: 2.1.7 | **Practice Name**: Implement Single Sign-On

***Description:*** The organization has implemented Single Sign-on (SSO), which is a authentication method that enables users to securely authenticate with multiple applications and websites by using one set of credentials. The user is then signed in to other applications automatically, regardless of the platform, technology, or domain the user is using.

***Implemented when:***

- The organization has implemented the 'Implement Authentication Protocol' (2.1.4) practice.
- The organization has implemented the Single Sign-on (SSO) authentication method.

Literature: (De (2017); (Auth0, 2021b; Onelogin, 2020)

Security — Authorization

| **Practice Code**: 2.2.2 | **Practice Name**: Implement Access Control |
|---|---|
| ***Description:*** The organization has implemented an access control method in order to identify and authorize consumer potential users of their API(s). In order to accomplish this, the Role-based Access Control (RBAC) method may be used, with which permissions may be assigned to users based on their role within the organization. Alternatively, the Attribute-based Access Control (ABAC) may be used, with which permissions are granted based on an identities' attributes. Optionally, RBAC and ABAC policies may be expressed by using the eXtensible Access Control Markup Language (XACML). | |
| ***Implemented when:*** • The organization has implemented the Role-based Access Control (RBAC) or Attribute-based Access Control (ABAC) method. | |
| Literature: (De (2017); Hofman and Rajagopal (2014); Thielens (2013); (Wikipedia, 2021) | |

| **Practice Code**: 2.2.4 | **Practice Name**: Implement Token Management |
|---|---|
| ***Description:*** The organization provides consumers of their API(s) with the ability to perform (access) token and API key management. This is an activity that involves measures to manage (i.e. review, store, create and delete) the tokens and API keys that are required to invoke back-end APIs. | |
| ***Implemented when:*** • The organization allows consumers to manage their tokens and API keys. | |
| Literature: (De, 2017; Hofman & Rajagopal, 2014) | |

| **Practice Code**: 2.2.6 | **Practice Name**: Implement Standardized Authorization Protocol |
|---|---|
| ***Description:*** The organization has implemented an industry-standardized authorization protocol, such as the OAuth 2.0 Authorization protocol. OAuth is used as a mechanism to provide authorization to a third-party application for access to an end user resource on behalf of them. OAuth helps with granting authorization without the need to share user credentials. | |
| ***Implemented when:*** • The organization has an industry-standardized authorization protocol. | |
| Literature: (De, 2017; Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015; Hofman & Rajagopal, 2014; Hohenstein, Zillner, & Biesdorf, 2018; Matsumoto, Kawai, & Takeda, 2017; Patni, 2017; Thielens, 2013; Xu, Jin, & Kim, 2019; Zhao et al., 2018) | |

| **Practice Code**: 2.2.7 | **Practice Name**: Implement Authorization Scopes |
|---|---|
| ***Description:*** The organization has implemented an authorization scopes mechanism, such as the OAuth 2.0 Scopes mechanism (Auth0, 2021a), to limit access to their application(s) to their users' accounts. An application can request one or more scopes, where after this information is then presented to the user in a consent screen. Then, the access token that was issued to the application will be limited to the scopes granted. | |
| ***Implemented when:*** • The organization has an authorization scopes mechanism. | |
| Literature: None. | |

Security — Threat Detection & Protection

**Practice Code**: 2.3.1 | **Practice Name**: Implement Allow & Deny IP Address Lists

***Description:*** The organization has the ability to impose allow and deny list policies. Through these policies, specific IPs can either be excluded from requests, or separate quotas can be given to internal users by throttling access depending on their IP address or address range.

***Implemented when:***

- The organization has the ability to impose allow and deny list policies.

Literature: (Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015; Hohenstein et al., 2018)

**Practice Code**: 2.3.2 | **Practice Name**: Implement Injection Threat Protection Policies

***Description:*** The organization has implemented injection threat protection security policies. Injection threats are common forms of attacks, in which attackers try to inject malicious code that, if executed on the server, can divulge sensitive information. These attacks may take the form of XML and JSON bombs or SQL and script injection.

***Implemented when:***

- The organization has injection threat policies in place against XML or JSON bombs or SQL or script injection.

Literature: (De (2017); Preibisch (2018); (OWASP, 2021a)

**Practice Code**: 2.3.5 | **Practice Name**: Implement DoS Protection

***Description:*** The organization has protection against DoS attacks in place. Hackers may try to bring down back-end systems by pumping unexpectedly high traffic through the APIs. Denial-of-service (DoS) attacks are very common on APIs. Hence, the organization should be able to detect and stop such attacks. Identification of a DoS attack is done through Spike Arrest.

***Implemented when:***

- The organization has protection against DoS attacks in place.

Literature: (De, 2017; Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015)

**Practice Code**: 2.3.7 | **Practice Name**: Implement Security Breach Protocol

***Description:*** The organization has a security breach protocol in place, which details what steps should be taken in the event where a security breach occurs. This protocol may include activities such as notifying stakeholders and consumers of the API, identifying the source of the breach by scanning activity logs, containing the breach by stopping the data leakage, and consulting third-party IT security and legal advice providers.

***Implemented when:***

- The organization has a security breach protocol in place.

Literature: (Reynold, 2020; Soliya, 2020)

**Practice Code**: 2.3.9 | **Practice Name**: Conduct Security Review

***Description:*** The organization has the ability to conduct security reviews that potential consumers of their API(s) must pass before being allowed to integrate the organization's API(s) into their application. This typically involves testing the degree to which customer data is protected and encrypted, and identifying security vulnerabilities that may be exploited, such as threats related to script injections and non-secure authentication and access control protocols.

***Implemented when:***

- The organization has the ability to conduct security reviews.

Literature: (Salesforce, 2020)

**Practice Code**: 2.3.10 | **Practice Name**: Implement Zero Trust Network Access (ZTNA)

***Description:*** The organization has implemented a Zero Trust Network Access (ZTNA) security architecture, where only traffic from authenticated users, devices, and applications is granted access to other users, devices, and applications within an organization. ZTNA may be regarded as a fine-grained approach to network access control (NAC), identity access management (IAM) and privilege access management (PAM), offering a replacement for VPN architectures. Optionally, a ZTNA may be implemented through third-party providers such as Akamai, Cloudflare, or Cisco.

***Implemented when:***

- The organization has implemented a Zero Trust Network Access (ZTNA) security architecture.

Literature: (Wikipedia, 2020)

| | | |
|---|---|---|
| Security | Encryption | **Practice Code**: 2.4.1 \| **Practice Name**: Implement Transport Layer Encryption |
| | | ***Description:*** The organization has implemented current and up-to-date encryption protocols such as Transport Layer Security (TLS). It is always desirable to have TLS compliant endpoints to safeguard against man-in-middle attacks, and bi-directional encryption of message data to protect against tampering. |
| | | ***Implemented when:*** • The organization has implemented TLS encryption. |
| | | Literature: (De, 2017; Familiar, 2015; Gadge & Kotwani, n.d.; Hofman & Rajagopal, 2014; Preibisch, 2018) |
| | | **Practice Code**: 2.4.2 \| **Practice Name**: Prevent Sensitive Data Exposure |
| | | ***Description:*** The organization has measures in place to prevent exposure of sensitive data, such as passwords and credit card numbers. Considering APIs expose data that may be sensitive, such data should be visible only to its intended recipient. If such data gets logged anywhere, it must be masked, encrypted or hashed at rest, which may be considered to be a data privacy requirement. Furthermore, caching for responses that contain sensitive data should be disabled, and all data in transit should be encrypted with protocols such as TLS. |
| | | ***Implemented when:*** • The organization has measures in place to prevent exposure of sensitive data. |
| | | Literature: (De (2017); (OWASP, 2021b) |
| | | **Practice Code**: 2.4.3 \| **Practice Name**: Implement Certificate Management |
| | | ***Description:*** The organization has the ability to manage its TLS certificates. This involves monitoring and managing the certificates' acquisition and deployment, tracking renewal, usage, and expiration of SSL/TLS certificates. |
| | | ***Implemented when:*** • The organization has the ability to manage its TLS certificates. |
| | | Literature: (De, 2017; Gadge & Kotwani, n.d.; Hohenstein et al., 2018; Siné, Haezebrouckl, & Emonet, 2015; Thielens, 2013) |

Performance — Resource Management

**Practice Code**: 3.1.2 | **Practice Name**: Implement Load Balancing

***Description:*** The organization has implemented load balancing to distribute API traffic to the back-end services. Various load balancing algorithms may be supported. Based on the selected algorithm, the requests must be routed to the appropriate resource that is hosting the API. Load balancing also improves the overall performance of the API.

***Implemented when:***

- The organization has implemented load balancing.

Literature: (Biehl, 2015; Ciavotta, Alge, Menato, Rovere, & Pedrazzoli, 2017; De, 2017; Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015; Montesi & Weber, 2016; Nakamura, 2017; Xu et al., 2019; Zhao et al., 2018)

**Practice Code**: 3.1.5 | **Practice Name**: Implement Scaling

***Description:*** The organization has the ability to scale the amount of available resources up or down depending on traffic and API usage in a reactive manner. This may be done either manually or automatically, through the use of a load balancer.

***Implemented when:***

- The organization has implemented the 'Implement Load Balancing' (3.1.2) practice.
- The organization has the ability to scale the amount of available resources up or down.

Literature: (Akbulut & Perros, 2019; Gadge & Kotwani, n.d.; Hofman & Rajagopal, 2014; Jacobson et al., 2011)

**Practice Code**: 3.1.6 | **Practice Name**: Implement Failover Policies

***Description:*** The organization has the ability to mitigate outages through the implementation of failover policies. This may be done by automatically deploying a service to a standby data center if the primary system fails, or is shut down for servicing. By being able to perform a failover, the particular service is guaranteed to be operational at one of the data centers. This is an extremely important function for critical systems that require always-on accessibility.

***Implemented when:***

- The organization is able to impose timeout policies on their API(s).

Literature: (Barracuda, 2020)

**Practice Code**: 3.1.10 | **Practice Name**: Implement Predictive Scaling

***Description:*** The organization has the ability to scale the amount of available resources up or down depending on traffic and API usage in a proactive manner. This may be done either manually or automatically, through the use of a load balancer as based on insights gained from predictive analytics.

***Implemented when:***

- The organization has implemented the 'Implement Load Balancing' (3.1.2) and 'Enable Predictive Analytics' (4.3.9) practices.
- The organization has implemented predictive scaling.

Literature: None.

Performance — Traffic Management

**Practice Code**: 3.2.1 | **Practice Name**: Set Timeout Policies

***Description:*** The organization is able to set timeout policies, by detecting and customizing the amount of time that is allowed to pass before a connection times out and is closed. Using timeout policies, the organization is able to ensure that the API always responds within a given amount of time, even if a long-running process hangs. This is important in high-availability systems where response performance is crucial so errors can be dealt with cleanly.

***Implemented when:***

- The organization is able to set timeout policies on their API(s).

Literature: (Tyk, 2020)

**Practice Code**: 3.2.2 | **Practice Name**: Implement Request Caching

***Description:*** The organization utilizes caching as a mechanism to optimize performance. As consumers of the API make requests on the same URI, the cached response can be used to respond instead of forwarding those requests to the back-end server. Thus caching can help to improve an APIs performance through reduced latency and network traffic.

***Implemented when:***

- The organization utilizes caching as a mechanism to optimize performance.

Literature: (Biehl, 2015; De, 2017; Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015; Hofman & Rajagopal, 2014; Indrasiri & Siriwardena, 2018; Patni, 2017; Preibisch, 2018; Šnuderl, 2018; Vijayakumar, 2018; Zhao et al., 2018)

**Practice Code**: 3.2.3 | **Practice Name**: Perform Request Rate Limiting

***Description:*** The organization has a mechanism in place with which limits on the amount of requests API consumers are allowed to make, may be imposed. Requests made within the specified limit are routed successfully to the target system. Those beyond the limit are rejected.

***Implemented when:***

- The organization has a rate limiting mechanism in place for their API(s).

Literature: (De, 2017; Gadge & Kotwani, n.d.; Gámez Díaz et al., 2015; Hofman & Rajagopal, 2014; Jacobson et al., 2011; Jayathilaka et al., 2015; Lourenço Marcos & Puccinelli de Oliveira, 2019; Raivio, Luukkainen, & Seppala, 2011; Šnuderl, 2018)

**Practice Code**: 3.2.4 | **Practice Name**: Perform Request Rate Throttling

***Description:*** The organization has a mechanism in place with which API requests may be throttled down, without the connection being closed. This can help to improve the overall performance and reduce impacts during peak hours. It helps to ensure that the API infrastructure is not slowed down by high volumes of requests from a certain group of customers or apps.

***Implemented when:***

- The organization has a rate throttling mechanism in place for their API(s).

Literature: (De, 2017; Familiar, 2015; Fremantle, Kopeckỳ, & Aziz, 2015; Gadge & Kotwani, n.d.; Hohenstein et al., 2018; Indrasiri & Siriwardena, 2018; Jacobson et al., 2011; Thielens, 2013; L. A. Weir et al., 2015)

**Practice Code**: 3.2.5 | **Practice Name**: Manage Quota

***Description:*** The organization has policies in place regarding the number of API calls that an app is allowed to make to the back end over a given time interval. Calls exceeding the quota limit may be throttled or halted. The quota allowed for an app depends on the business policy and monetization model of the API. A common purpose for a quota is to divide developers into categories, each of which has a different quota and thus a different relationship with the API.

***Implemented when:***

- The organization has implemented the 'Perform Request Rate Limiting' (3.2.3) practice or 'Perform Request Rate Throttling' (3.2.4) practice.
- The organization has quota policies for their API(s) in place.

Literature: (De, 2017)

**Practice Code**: 3.2.6 | **Practice Name**: Apply Data Volume Limits

***Description:*** The organization has a mechanism in place with which the amount of data consumers of their API(s) are allowed to consume in one call may be limited. This can help to improve the overall performance and reduce impacts during peak hours. It helps to ensure that the API infrastructure is not slowed down by calls that transport unnecessarily high chunks of data volumes.

***Implemented when:***

- The organization has implemented the 'Monitor Resource Usage' (4.1.5) practice.
- The organization has a data volume limiting mechanism in place.

Literature: (Dropbox, 2021)

**Practice Code**: 3.2.9 | **Practice Name**: Prioritize Traffic

***Description:*** The organization is able to give a higher priority in terms of processing API calls, based on certain customer characteristics and/or classes. This priority may be based on their subscription, customer relationships, or agreements made in the SLA.

***Implemented when:***

- The organization is able to prioritize traffic based on customer characteristics and/classes.

Literature: (De, 2017)

Observability — Monitoring

**Practice Code**: 4.1.1 | **Practice Name**: Monitor API Health

***Description:*** The organization is able to perform health monitoring on its API(s), possibly through an management platform, external monitoring tool/dashboard, functional testing or custom scripts and plugins. This should return basic information such as the operational status of the API, indicating its ability to connect to dependent services.

***Implemented when:***

- The organization is able to perform health monitoring on its API(s).

Literature: (Averdunk and Moen (2020); (Gadge & Kotwani, n.d.)

**Practice Code**: 4.1.3 | **Practice Name**: Monitor API Performance

***Description:*** The organization is able to perform performance monitoring on its API(s), possibly through an management platform, external monitoring tool/dashboard, functional testing or custom scripts and plugins. Doing so should provide performance statistics that track the latency within the platform and the latency for back-end calls. This helps the organization in finding the source of any performance issues reported on any API.

***Implemented when:***

- The organization is able to perform performance monitoring on its API(s).

Literature: (De, 2017; Xu et al., 2019)

**Practice Code**: 4.1.5 | **Practice Name**: Monitor Resource Usage

***Description:*** The organization is able to perform resource monitoring on its API(s), possibly through an management platform, external monitoring tool/dashboard, functional testing or custom scripts and plugins. Doing so should provide hardware metrics related to the resources that are consumed as a result of calls made to the API(s), such as CPU, disk, memory, and network usage.

***Implemented when:***

- The organization is able to perform resource monitoring on its API(s).

Literature: (Kubernetes, 2021)

Observability — Logging

**Practice Code**: 4.2.1 | **Practice Name**: Log Errors

***Description:*** The organization has the ability to internally log errors that are generated as a result of consumption of their APIs. Error logs should typically contain fields that capture information such as the date and time the error has occurred, the error code, and the client IP and port numbers.

***Implemented when:***

- The organization has the ability to internally log errors.

Literature: (Andrey Kolychev, 2019; De, 2017; Medjaoui et al., 2018)

**Practice Code**: 4.2.2 | **Practice Name**: Log Access Attempts

***Description:*** The organization has the ability to generate access logs, in which HTTP requests/responses are logged, to monitor the activities related to an APIs usage. Access logs offer insight into who has accessed the API, by including information such as the consumer's IP address.

***Implemented when:***

- The organization is able to perform access logging.

Literature: (WSO2, 2020)

**Practice Code**: 4.2.3 | **Practice Name**: Log Activity

***Description:*** The organization has the ability to perform basic logging of API activity, such as access, consumption, performance, and any exceptions. In doing so, it may be determined what initiated various actions to allow for troubleshooting any errors that occur.

***Implemented when:***

- The organization is able to perform activity logging.

Literature: (De, 2017; Fremantle et al., 2015; Gadge & Kotwani, n.d.)

**Practice Code**: 4.2.5 | **Practice Name**: Audit User Activity

***Description:*** The organization is able to perform user auditing. Doing so enables the organization to review historical information regarding API activity, to analyze who accesses an API, when it is accessed, how it is used, and how many calls are made from the various consumers of the API.

***Implemented when:***

- The organization is able to perform user auditing.

Literature: (De, 2017; Gadge & Kotwani, n.d.)

Observability — Analytics

**Practice Code**: 4.3.2 | **Practice Name**: Report Errors

***Description:*** The organization has the ability to report any errors to consumers that may occur during usage of their API(s). Error reports typically include information such as the error code and text describing why the error has occurred.

***Implemented when:***

- The organization has implemented the 'Log Errors' (4.2.1) practice.
- The organization is able to report any errors to consumers.

Literature: (Andrey Kolychev, 2019; De, 2017; Medjaoui et al., 2018)

**Practice Code**: 4.3.3 | **Practice Name**: Broadcast API Status

***Description:*** The organization broadcasts the status of its API(s) to consumers by providing them with operational information on the API in the form of an external status page, possibly on the developer portal or a website. The function of this status page is to let consumers know what is going on with the API at a technical level at any point in time.

***Implemented when:***

- The organization has implemented the 'Monitor API Health' (4.1.1) practice.
- The organization broadcasts the operational status of its API(s) to consumers.

Literature: (Sandoval, 2018c)

**Practice Code**: 4.3.6 | **Practice Name**: Generate Custom Analysis Reports

***Description:*** The organization is able to generate custom analysis reports on metrics of choice, possibly through an API management platform or monitoring tool.

***Implemented when:***

- The organization is able to generate custom analysis reports.

Literature: (De, 2017)

**Practice Code**: 4.3.7 | **Practice Name**: Set Alerts

***Description:*** The organization has the ability to set and configure alerts that should trigger in case of certain events or thresholds being exceeded. Such events or thresholds may include resource limits being exceeded, or occurrence of outages. Ideally, the organization is able to configure what persons should be alerted about the event, and through what communication channel they should be contacted.

***Implemented when:***

- The organization has implemented the 'Monitor API Health' (4.1.1), 'Monitor API Performance' (4.1.3), and 'Monitor API Resource Usage' (4.1.5) practices.
- The organization has the ability to set and configure alerts.

Literature: (Uptrends, 2021)

**Practice Code**: 4.3.9 | **Practice Name**: Enable Predictive Analytics

***Description:*** The organization has the ability to aggregate predictive analytics, through techniques such as pattern recognition, data mining, predictive modelling, or machine learning, by analyzing current and historical facts to make predictions about future or otherwise unknown events.

***Implemented when:***

- The organization has implemented the 'Monitor API Performance' (4.1.3) and 'Monitor API Resource Usage' (4.1.5) practices.
- The organization has the ability to aggregate predictive analytics.

Literature: None.

Community | Developer Onboarding

**Practice Code**: 5.1.1 | **Practice Name**: Facilitate Developer Registration

***Description:*** The organization has a mechanism in place with which API consumers are able to register to the API so that they can obtain access credentials. Consumers can then select an API and register their apps to use it.

***Implemented when:***

- The organization has a mechanism in place with which API consumers are able to register to their API(s).

Literature: (De, 2017)

**Practice Code**: 5.1.4 | **Practice Name**: Provide SDK Support

***Description:*** The organization offers API consumers the option to either download client-side SDKs for the API, or generate the SDK themselves from standard API definition formats such as OpenAPI (formerly known as Swagger). These functionalities are usually offered through the developer portal, where app developers often look for device-specific libraries to interact with the services exposed by the API.

***Implemented when:***

- The organization offers API consumers the option to download or generate client-side SDKs for their API(s).

Literature: (De, 2017)

**Practice Code**: 5.1.5 | **Practice Name**: Implement Interactive API Console

***Description:*** The organization provides API consumers with an interactive console. Using this console, developers are able to test the behavior of an API.

***Implemented when:***

- The organization provides API consumers with an interactive console.

Literature: (Biehl, 2015)

**Practice Code**: 5.1.8 | **Practice Name**: Provide Sandbox Environment Support

***Description:*** The organization provides API consumers with an environment that they can use to mimic the characteristics of the production environment and create simulated responses from all APIs the application relies on.

***Implemented when:***

- The organization provides API consumers with a sandbox environment.

Literature: (Bui (2018); Jacobson et al. (2011); Patni (2017); (Mueller, 2020)

Community | Support

**Practice Code**: 5.2.1 | **Practice Name**: Establish Communication Channel

***Description:*** The organization has established a communication channel between the API provider and consumer with which support may be provided to the consumer. Possible communication media include email, phone, form, web, community forum, blogs or the developer portal.

***Implemented when:***

- The organization has established one of the following communication channels with consumers of their API(s): email/phone/form/web/ community forum/blog/developer portal.

Literature: (De, 2017; Jacobson et al., 2011)

**Practice Code**: 5.2.4 | **Practice Name**: Manage Support Issues

***Description:*** The organization is able to manage any support issues with their API(s). API consumers must be able to report any issues, bugs or shortcomings related to the API. They should be able to raise support tickets and seek help regarding API usage. Additionally, the API provider must be able to track and prioritize support tickets.

***Implemented when:***

- The organization is able to manage any support issues with their API(s).

Literature: (De, 2017; Jacobson et al., 2011)

**Practice Code**: 5.2.6 | **Practice Name**: Dedicate Developer Support Team

***Description:*** The organization employs a dedicated that offers support to consumers of their API(s). This team should be well-trained and possess knowledge that enables them to assist consumers with any problems or difficulties they may experience during the usage or implementation of the API.

***Implemented when:***

- The organization has implemented the 'Establish Communication Channel' (5.2.1) practice.
- The organization employs a dedicated developer team that offers support to consumers of their API(s).

Literature: None.

Community | Documentation

**Practice Code**: 5.3.1 | **Practice Name**: Use Standard for Reference Documentation

***Description:*** The organization provides consumers of their API(s) with basic reference documentation on their website, developer portal or an external, third-party documentation platform. This documentation should document every API call, every parameter, and every result so that consumers are informed on the API's functionality. Additionally, it must be specified using a documentation framework such as Swagger, RAML, API Blueprint, WADL, Mashery ioDocs, Doxygen, ASP.NET API Explorer, Apigee Console To-Go, Enunciate, Miredot, Dexy, Docco or TurnAPI.

***Implemented when:***

- The organization provides consumers of their API(s) with basic reference documentation.
- The organization utilizes one of the following (or comparable) documentation tools to specify its API documentation: Swagger (OpenAPI), RAML, API Blueprint, WADL, Mashery ioDocs, Doxygen, ASP.NET API Explorer, Apigee Console To-Go, Enunciate, Miredot, Dexy, Docco or TurnAPI.

Literature: (De, 2017; Jacobson et al., 2011; Medjaoui et al., 2018)

**Practice Code**: 5.3.3 | **Practice Name**: Provide Start-up Documentation & Code Samples

***Description:*** The organization provides consumers of their API(s) with start-up documentation on on their website, developer portal or an external, third-party documentation platform. This type of documentation explains key concepts by summarizing the reference documentation, accelerating understanding as a result. Optionally, a list of Frequently Asked Questions and code samples that may be readily used in apps to invoke the API may be included.

***Implemented when:***

- The organization has implemented the 'Use Standard for Reference Documentation' (5.3.1) practice.
- The organization provides consumers of their API(s) with start-up documentation.

Literature: (De, 2017; Jacobson et al., 2011)

**Practice Code**: 5.3.5 | **Practice Name**: Create Video Tutorials

***Description:*** The organization is able to create video tutorials in order to provide consumers with visual information that details how to use the API and integrate it into their applications.

***Implemented when:***

- The organization is able to create video tutorials.

Literature: None.

Community — Community Engagement

**Practice Code**: 5.4.1 | **Practice Name**: Maintain Social Media Presence

***Description:*** The organization is able to maintain their social media presence on platforms such as Facebook or Twitter. This may involve activities such as reporting on the API's status, announcing news and updates, responding to questions, or reacting to feedback.

***Implemented when:***

- The organization is able to maintain their social media presence on platforms such as Facebook or Twitter.

Literature: None.

**Practice Code**: 5.4.3 | **Practice Name**: Provide Community Forum

***Description:*** The organization provides (potential) consumers of their API(s) with a community forum, possibly through a website or API management platform. This forum may assist in building and interconnecting a developer community, by providing them with a central hub they can use to communicate with one another and the organization. Additionally, it may serve as a repository with guides on API usage, documentation and support.

***Implemented when:***

- The organization provides API consumers with a community forum.

Literature: (De, 2017)

**Practice Code**: 5.4.4 | **Practice Name**: Provide Developer Portal

***Description:*** The organization provides (potential) consumers of their API(s) with a developer portal. A developer portal provides the platform for an API provider to communicate with the developer community. Addtionally, it typically offers functionality such as user registration and login, user management, documentation, API key management, test console and dashboards.

***Implemented when:***

- The organization has implemented a developer portal.

Literature: (De, 2017; Fremantle et al., 2015; Medjaoui et al., 2018; Siné et al., 2015)

**Practice Code**: 5.4.7 | **Practice Name**: Organize Events

***Description:*** The organization is actively involved in organizing or participating in events that are aimed towards engaging and motivating the developer community to incorporate their API(s) into their applications. This may include events such as hackathons, conferences, or workshops.

***Implemented when:***

- The organization is actively involved in organizing or participating in developer community events.

Literature: None.

**Practice Code**: 5.4.9 | **Practice Name**: Dedicate Evangelist

***Description:*** The organization employs a dedicated API evangelist. This individual is responsible for evangelizing the API by gathering consumer feedback, and promoting the organization's API(s) by creating samples, demos, training materials and performing other support activities aimed towards maximizing the developer experience.

***Implemented when:***

- The organization employs a dedicated API evangelist.

Literature: None.

Community — Portfolio Management

| **Practice Code**: 5.5.1 | **Practice Name**: Enable API Discovery |
|---|---|
| ***Description:*** The organization provides potential consumers of their API(s) with a mechanism to obtain information, such as documentation and metadata, about their API(s). This mechanism may take the shape of an external website, hub or repository that consumers can freely browse through. | |
| ***Implemented when:*** • The organization has a mechanism in place with which their API(s) may be discovered. | |
| Literature: (Biehl, 2015; Hofman & Rajagopal, 2014) | |

| **Practice Code**: 5.5.4 | **Practice Name**: Provide API Catalog |
|---|---|
| ***Description:*** The organization provides API consumers with an API Catalog. This is a a searchable catalog of APIs. An API catalog is also sometimes referred to as an API registry. API consumers should be able to search the catalog based on various metadata and tags. The catalog should document the API functionality, its interface, start-up documentation, terms and conditions, reference documentation, and so forth. | |
| ***Implemented when:*** • The organization has implemented the 'Enable API Discovery' (5.5.1) practice. • The organization provides API consumers with a searchable API catalog. | |
| Literature: (De, 2017; Hofman & Rajagopal, 2014; Lourenço Marcos & Puccinelli de Oliveira, 2019; Medjaoui et al., 2018; Vijayakumar, 2018) | |

| **Practice Code**: 5.5.5 | **Practice Name**: Bundle APIs |
|---|---|
| ***Description:*** The organization is able to combine two or more APIs into a bundle. This is a collection of API products that is presented to developers as a group, and typically associated with one or more rate plans for monetization. | |
| ***Implemented when:*** • The organization is able to combine two or more APIs into a bundle. | |
| Literature: (Apigee, 2020) | |

Commercial — Service-Level Agreements

**Practice Code**: 6.1.1 | **Practice Name**: Publish Informal SLA

***Description:*** The organization has the ability to publish and agree upon an informal, bare-bones SLA with consumers of their API(s). This type of SLA is minimalistic and loose in terms of the nature and amount of agreements it contains, as well as the consequences attached to these agreements should they be violated. This type of SLA is satisfactory for organizations that provide non-critical services and that have close relationships with their consumers and partners.

***Implemented when:***

- The organization has the ability to publish and agree upon an informal SLA with consumers.

Literature: None.

**Practice Code**: 6.1.3 | **Practice Name**: Provide SLA

***Description:*** The organization has the ability to provide and agree upon a formal, elaborate SLA with consumers of their API(s). This type of SLA is extensive and strict in terms of the nature and amount of agreements it contains, as well as the consequences attached to these agreements should they be violated. Typically, agreements regarding the guaranteed uptime of the API on a monthly or yearly basis are included in this type of SLA, along with guaranteed response times in the event of incidents, as well as policies regarding privacy, security, and possibly rate and data quotas. Additionally, when providing a formal SLA, the organization should have a plan in place that details what course of action should be taken in the event where agreements are failed to be upheld.

***Implemented when:***

- The organization has the ability to provide and agree upon a formal SLA with consumers.

Literature: (De, 2017)

**Practice Code**: 6.1.6 | **Practice Name**: Proactively Monitor SLAs

***Description:*** The organization is able to proactively monitor metrics that are relevant in checking whether the agreements made with API consumers are adhered to. Such metrics may include availability, performance and functional correctness.

***Implemented when:***

- The organization has implemented the 'Monitor API Resource Usage' (4.1.5) practice.
- The organization is able to perform SLA monitoring.

Literature: (Moiz, 2020)

**Practice Code**: 6.1.7 | **Practice Name**: Customize Personalized SLA

***Description:*** The organization has the ability to provide consumers of their API(s) with personalized SLAs. This type of SLA is suitable for intensive consumers that utilize services offered by the API in such a way that requires customized agreements as compared to those that are offered as part of the organization's standard SLA. For example, some consumers may require minimal latency and response times for their calls, want to make large amounts of calls, or demand API uptime approaching 100%. Additionally, a personalized SLA may be required due to the consumer being located in a different geographic location than other consumers, requiring customized agreements with regards to privacy laws and regulations.

***Implemented when:***

- The organization has implemented the 'Provide SLA' (6.1.3) practice.
- The organization has the ability to provide consumers of their API(s) with personalized SLAs.

Literature: (Hosting Manual, 2020)

Commercial — Monetization Strategy

**Practice Code**: 6.2.6 | **Practice Name**: Adopt Subscription-based Monetization Model

***Description:*** The organization has adopted a monetization model that is based on a subscription basis. With this model, API consumers pay a flat monthly fee and are allowed to make a certain number of API calls per month.

***Implemented when:***

- The organization has implemented the 'Implement Subscription Management System' (6.3.2) and 'Manage Quota' (3.2.5) practices.
- The organization has adopted a monetization model that is based on a subscription basis.

Literature: (Budzynski, 2016)

**Practice Code**: 6.2.8 | **Practice Name**: Adopt Tier-Based Monetization Model

***Description:*** The organization has adopted a monetization model that is based on tiered access. Typically, each tier has its own set of services and allowances for access to API resources, with increasing prices for higher tiers.

***Implemented when:***

- The organization has implemented the 'Prioritize Traffic' (3.2.7) and 'Manage Quota' (3.2.5) practices.
- The organization utilizes a monetization model that is based on tiered access.

Literature: (Budzynski, 2016; Redhat, 2020)

**Practice Code**: 6.2.9 | **Practice Name**: Adopt Freemium Monetization Model

***Description:*** The organization has adopted a monetization model that is based on freemium functionalities and access. This involves providing consumers with a limited part of the services and functionalities the API offers as a whole. Consumers that wish to utilize all services and functionalities are required to have an active, paid subscription to the API.

***Implemented when:***

- The organization utilizes a monetization model that is based on freemium functionalities and access.

Literature: (Budzynski, 2016; Redhat, 2020)

**Practice Code**: 6.2.10 | **Practice Name**: Adopt Metering-Based Monetization Model

***Description:*** The organization utilizes a monetization model that is based on metering. With this model, API consumers pay for the amount of resources they use. This may be measured in terms of bandwidth, storage or amount of calls made.

***Implemented when:***

- The organization has implemented the 'Monitor Resource Usage' (4.1.5) practice.
- The organization utilizes a monetization model that is based on metering.

Literature: (Budzynski, 2016; Redhat, 2020)

Commercial — Account Management

| **Practice Code**: 6.3.2 | **Practice Name**: Implement Subscription Management System |
|---|---|
| ***Description:*** The organization has a system in place with which it is able to manage existing subscriptions (consumers of) on their API. A subscription management system provides support for billing on a recurring basis, as well as providing insight into active subscriptions. | |
| ***Implemented when:*** • The organization has implemented a subscription management system. | |
| Literature: (Fremantle et al., 2015; Preibisch, 2018; Raivio et al., 2011) | |

| **Practice Code**: 6.3.7 | **Practice Name**: Report on API Program Business Value |
|---|---|
| ***Description:*** The organization is able to generate business value reports associated with their API(s). Business value reports gauge the monetary value associated with the API program. Monetization reports of API usage provide information on the revenue generated from the API. Value-based reports should also be able to measure customer engagements. Engagements can be measured by the number of unique users, the number of developers registered, the number of active developers, the number of apps built using the APIs, the number of active apps, and many other items. Optionally, these metrics may be visualized in the form of dashboards, so that they may then easily be shared and presented to relevant internal stakeholders to communicate the API program's business value. | |
| ***Implemented when:*** • The organization has implemented the 'Generate Custom Analysis Reports' (4.3.6) practice. • The organization is able to generate business value reports associated with their API(s). | |
| Literature: (De, 2017) | |

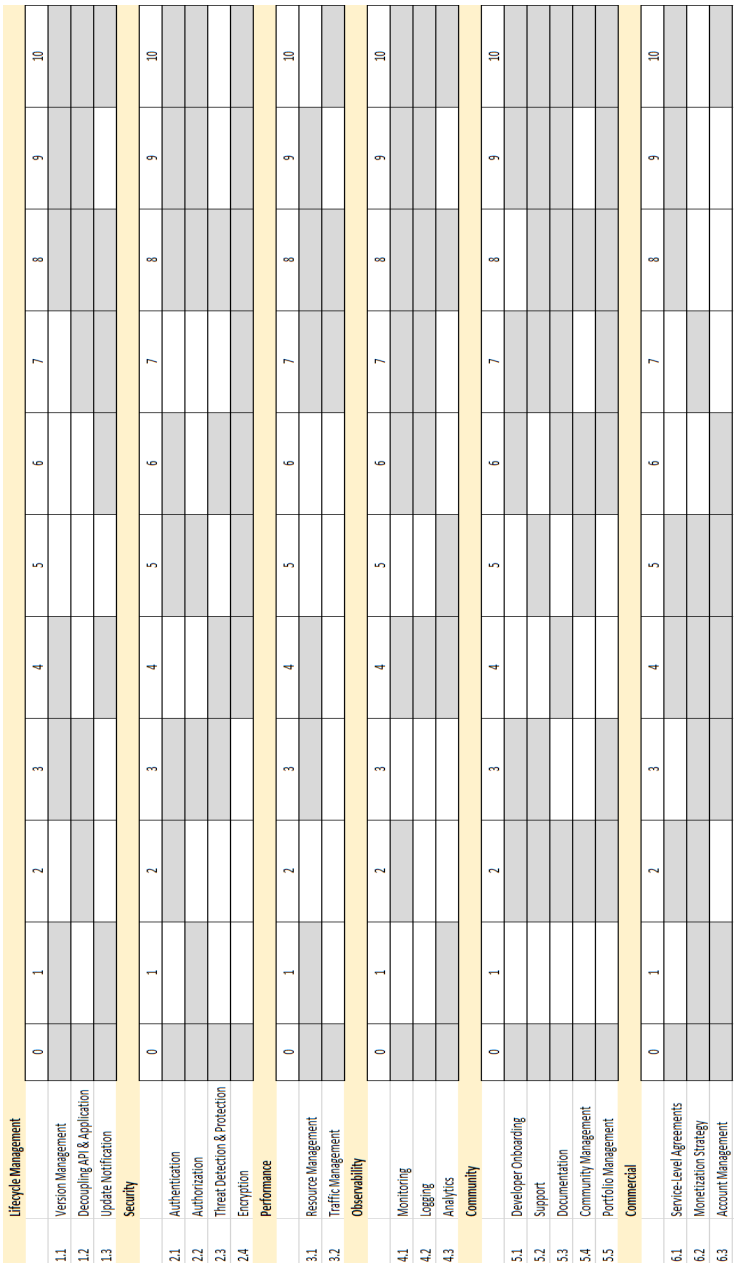
| **Practice Code**: 6.3.8 | **Practice Name**: Provide Subscription Report to Customer |
|---|---|
| ***Description:*** The organization is able to generate subscription reports for consumers of their API(s). These reports contain metrics gathered through internal monitoring and analytics. Such metrics may include amount of calls made, performance, and status regarding remaining allowed quotas. | |
| ***Implemented when:*** • The organization has implemented the 'Generate Custom Analysis Reports' (4.3.6) and 'Implement Subscription Management System' (6.3.2) practices. • The organization is able to generate subscription reports for consumers of their API(s). | |
| Literature: (De, 2017) | |

| **Practice Code**: 6.3.9 | **Practice Name**: Proactively Suggest Optimizations to Customers |
|---|---|
| ***Description:*** The organization has the ability to train and help customers in using their API(s) as well and efficiently as possible. This may be in the best interest of both parties, as optimizing inefficient calls may positively impact traffic load on the API infrastructure. | |
| ***Implemented when:*** • The organization has implemented the 'Monitor API Performance' (4.1.3) and 'Monitor Resource Usage' (4.1.5) practices. • The organization is able to generate business value reports. | |
| Literature: (Bui, 2018; De, 2017) | |

# Appendix E

# Case Study

## E.1 Data Collection Spreadsheet



| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Lifecycle Management** | | | | | | | | | | | | |
| 1.1 | Version Management | | | | | | | | | | | |
| 1.2 | Decoupling API & Application | | | | | | | | | | | |
| 1.3 | Update Notification | | | | | | | | | | | |
| **Security** | | | | | | | | | | | | |
| 2.1 | Authentication | | | | | | | | | | | |
| 2.2 | Authorization | | | | | | | | | | | |
| 2.3 | Threat Detection & Protection | | | | | | | | | | | |
| 2.4 | Encryption | | | | | | | | | | | |
| **Performance** | | | | | | | | | | | | |
| 3.1 | Resource Management | | | | | | | | | | | |
| 3.2 | Traffic Management | | | | | | | | | | | |
| **Observability** | | | | | | | | | | | | |
| 4.1 | Monitoring | | | | | | | | | | | |
| 4.2 | Logging | | | | | | | | | | | |
| 4.3 | Analytics | | | | | | | | | | | |
| **Community** | | | | | | | | | | | | |
| 5.1 | Developer Onboarding | | | | | | | | | | | |
| 5.2 | Support | | | | | | | | | | | |
| 5.3 | Documentation | | | | | | | | | | | |
| 5.4 | Community Management | | | | | | | | | | | |
| 5.5 | Portfolio Management | | | | | | | | | | | |
| **Commercial** | | | | | | | | | | | | |
| 6.1 | Service-Level Agreements | | | | | | | | | | | |
| 6.2 | Monetization Strategy | | | | | | | | | | | |
| 6.3 | Account Management | | | | | | | | | | | |

## E.2 Evaluation Survey

Thank you for having previously participated in providing valuable input and evaluating the first version of the Focus Area Maturity Model for API Management (API-m-FAMM). Now that the results of the expert interviews have been analyzed and processed, the API-m-FAMM has been modified to accommodate the variety of suggestions and improvements that were provided by experts. In order to measure whether the model is usable in practice without assistance, and whether these modifications have improved certain characteristics of the model, such as it's usefulness, effectiveness, and ease of use, we would like to ask you a few quick questions.

In order to review the API-m-FAMM in it's entirety, please refer to the visual variant of the model you have been provided through email. Should you have any further questions, please refer to the source document you have been provided with, or feel free to contact one of the researchers:

m.mathijssen@students.uu.nl
slinger.jansen@uu.nl
michiel.overeem@afas.nl

1. What is your e-mail address?
2. Have you filled out the provided spreadsheet or are you interested in doing so? (yes/no)

**Evaluation Questions**

1. For which organization or software product have you filled out the spreadsheet?
2. Do you want your answers to be anonymized, or are you comfortable with the name of your organization being published in our work along with the answers you have provided us? (I want the name of my organization to be anonymized / I am comfortable with the name of my organization being published)
3. Do you want us to contact you and discuss these answers with you, to clarify or explain parts of the model? (yes/no)
4. How easy did you find it to use the API-m-FAMM to self-assess and evaluate your organization's maturity in API management? ( 1 / Not easy at all - 5 / Very easy)
5. How effective do you think the API-m-FAMM is in helping you and your organization improve on their API management related processes? ( 1 / Not effective at all - 5 / Very effective)
6. How useful do you think the API-m-FAMM was in providing you and your organization with valuable and interesting insights in your organization's API management related processes? (1 / Not useful at all - 5 / Very useful)
7. Are you expecting that you or your organization will use the API-m-FAMM in practice again to evaluate and improve on your API management related

processes, so that you can track your progress? (1 / Not likely at all - 5 / Very likely)

8. Do you have any further questions, comments, or suggestions for improvement of the API-m-FAMM?

Thank you again for your participation in evaluating the API-m-FAMM! Once this project has been finalized, we will provide you with a copy of the final model and the thesis describing the project.