DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Data Engineering and Analytics

Identification and Evaluation of Incentive Mechanisms for Opening Internal Systems via Partner APIs

Irena Stoilova

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Data Engineering and Analytics

Identification and Evaluation of Incentive Mechanisms for Opening Internal Systems via Partner APIs

Identifikation und Bewertung von Anreizmechanismen zur Öffnung interner Systeme über Partner-APIs

Author:	Irena Stoilova
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Gloria Bondel
Submission Date:	15.07.2020

I confirm that this master's thesis in data engineering and analytics is my own work and I have documented all sources and material used.

Munich, 15.07.2020 Irena Stoilova

Acknowledgments

First and foremost I would like to thank the advisor of this thesis M.Sc. Gloria Bondel from the chair of Software Engineering for Business Information Systems of the Technical University of Munich for the great support and feedback during the writing of the thesis, for the discussions and the problem-solving attitude throughout the process. Further, I would like to thank Dr. Manoj Mahabaleshwar as an advisor from the industry partner side. I want to thank both for the constructive criticism as well as for the trust.

A special thank you to the supervisor of the thesis Prof. Dr. Florian Matthes for the given opportunity and the realization of the project with the industry partner.

Last, I would like to thank my life partner Boris Idesman and my family for the support and encouragement in the last few eventful months.

Abstract

In today's world the need of inter-connectivity between systems is gaining greater importance. An enabler of such connections are Application Programming Interfaces (APIs). APIs allow access to big volumes of data, services and processes. Companies value their strategic potential and are getting a better understanding of the benefits of building APIs. In today's economy organizations do not operate solely, but often collaborate with others to create a competitive product in the vastly changing business. By opening internal projects via APIs new partnerships are feasible and by doing so, hidden potentials of existing projects are realised.

Numerous valuable advantages of APIs in regards to collaborations have been studied. They provide a high level of data security and management of access. They bring agility, scalability and accessibility to big data systems. Further, their implementation allows for the developers of the API to track usage of the services they are providing.

However, many projects end with the development of a functionality for internal use and rarely the development of an API afterwards lies in the scope of the project. Thus, development teams and other key roles involved in the process need to be incentivized to create APIs. Goal of this thesis is to research and design a process for incentive mechanisms for opening internal systems via APIs.

The process will focus on defining and overcoming the challenges that exist when developing an API in internal solutions, e.g. big organizations. Further, existing incentive mechanisms will be gathered and examined in the context dependencies of a corporation. The resulting findings of this analysis will be then defined into a list of recommendations for incentivizing teams to provide partner APIs.

The proposed solution for motivating teams is adapted to the specifics of an organisation, but takes into account the requirements of interviewees from more than one project. Twenty one professionals with different roles like software engineers, architects, product owners and leading positions are interviewed. In total, nine recommendations of action based on related literature and semi-structured interviews are proposed. They span trough different aspects, such as providing guidelines and platforms and top management initiatives, and cover methods from demand to motivation encouragement.

The evaluation of the recommendations of action is done qualitatively by conducting interviews with experts from the field.

Kurzfassung

Heutzutage gewinnt die Notwendigkeit der Konnektivität zwischen Systemen zunehmend an Bedeutung. Solche Verbindungen sind mittels Application Programming Interfaces (APIs) möglich. APIs ermöglichen den Zugriff auf große Datenmengen, Dienste und Prozesse. Unternehmen sehen ihr strategisches Potenzial und haben ein besseres Verständnis für die Vorteile der Entwicklung von APIs. In der heutigen Wirtschaft, wo sich die Business Trends schnell ändern, arbeiten Organisationen nicht isoliert, sondern häufig mit anderen zusammen, um ein konkurrenzfähiges Produkt zu schaffen. Durch das Öffnen interner Systeme über APIs sind neue Partnerschaften möglich und auf diese Weise wird ihre Wertschöpfung erhöht.

Zahlreiche wertvolle Vorteile von APIs in Bezug auf die Zusammenarbeit wurden rescherschiert. Sie bieten ein hohes Maß an Datensicherheit und Zugriffsverwaltung. Sie bieten Flexibilität, Skalierbarkeit und Zugänglichkeit zu Big-Data-Systeme. Darüber hinaus ermöglicht ihre Implementierung den Entwicklern, die Nutzung der von ihnen bereitgestellten Dienste zu untersuchen.

Viele Projekte enden jedoch mit der Entwicklung einer Funktionalität für den internen Gebrauch und selten ist die Entwicklung einer API im Rahmen des Projekts im Voraus geplant. Daher müssen Entwicklungsteams, sowie andere am Prozess beteiligte Personen, dazu inzentiviert werden. Ziel dieser Studie ist es, einen Prozess für Anreizmechanismen zum Öffnen interner Systeme über APIs zu erforschen und zu entwerfen.

Der Prozess konzentriert sich auf die Definition und Überwindung der Herausforderungen, die bei der Entwicklung einer API in internen Lösungen, z.B. bei Großunternehmen, bestehen. Darüber hinaus werden bestehende Anreizmechanismen im Kontextabhängigkeiten eines Unternehmens gesammelt und untersucht. Die daraus resultierenden Ergebnisse dieser Analyse werden dann in Handlungsempfehlungen definiert, mit den Teams dazu angeregt werden, Partner-APIs bereitzustellen.

Die vorgeschlagene Lösung zur Motivation von Teams ist an die Besonderheiten einer Organisation angepasst, berücksichtigt jedoch die Anforderungen der Interview Partnern aus mehr als einem Projekt. Ein und zwanzig Fachkräfte mit unterschiedlichen Rollen wie Softwareentwickler, Architekten, Projekt Owner und Führungskräfte werden interviewiert. Insgesamt werden neun Handlungsempfehlungen vorgeschlagen, die auf Literaturreschersche und halbstrukturierten Interviews beruhen. Sie umfassen verschiedene Aspekte, wie z.B. die Verfolgung von Richtlinien und das Anbieten von Plattformen sowie Initiativen des Top Managements. Sie decken Methoden von der Anforderung bis zur Motivationsförderung ab. Die Auswertung der Handlungsempfehlungen erfolgt qualitativ durch Interviews mit Experten aus der Praxis.

Contents

1. Introduction

1.1. Motivation

In today's digital economy a strategic application programming interface (API) management is gaining importance for successful organisations. To fully use the potential value of APIs, a thorough understanding of their usage, potential business models and monetization strategies should be present [1]. Many leading companies in the API economy, such as Google with Apigee, Amazon Web Service with Amazon API Gateway and others, have created API management platforms that assist them, their partners and their clients to establish a platform for communication and consumption and also usage analysis [2].

No organisation can prosper innovatively without the engagement of its teams and professionals. The way to achieve a new API strategy for a large company spans through various methods for the engagement and motivation for providing APIs. The famous statement of Amazon's CEO Jeff Bezos pressures employees to provide access to functionality explicitly with interfaces, that further need to be well documented [3], [4]. These requirements are often seen as the success that followed with Amazon Web Technologies. The fact that the services provided there have been first widely implemented in Amazon itself, contributed to their well-designed functionality and met the real needs of professionals worldwide.

The goal of this study is to research the wide range of possible incentive mechanisms for opening internal systems via partner APIs. It takes into account motivators spanning from a direct demand as described above to intrinsic motivators. Our research further investigates the differences that could be present for the various roles involved in the API economy – developers, architects, product owners and upper management. They have different perspectives on why the establishment of a partnership is valuable to them. The abilities that APIs give to an organisation advance with the development of technologies. Cloud Computing, Big Data, Artificial Intelligence and Deep Learning, Internet of Things (IoT) are topics that are vital for companies to enhance existing systems into innovative digital business solutions [5], [6], [7].

The different types of APIs bring various benefits depending on the systems, companies and business they connect. Perhaps the most important step for any organisation, when identifying its role in the API economy, is locating potential sources of API value and placing those on within API value chain [1]. The experience gathered with APIs in internal settings is valuable and is a starting point for many teams, that want to be able to open their systems to consumers. Partner APIs have multiple definitions in literature [2], [8], [9]. One differentiation of interfaces is done based in on the business value that they represent. Thus, they are divided into APIs for system connectivity, APIs for enterprise mobility and productivity, business-to-customer APIs, APIs for partner collaboration and APIs for revenue generation [8]. Within a large organisation any of those APIs can be of interest to the context of this thesis since each and every one of them holds its obstacles and requires different incentives.

1.2. Research Objectives

This thesis intents to study challenges and incentives for creating and maintaining APIs in large organisations and further for the opening of internal systems via partner APIs. The research topic is divided into the following four research questions (RQs). Their formulation and a more detailed explanation of the objective of the question is to be found below:

> **RQ1: What are the challenges for providing partner APIs in internal solutions?**

With this question the identification of challenges, that different roles see for the providing of partner APIs is desired. Partner APIs technically enable business-to-business relationships. Towards the answering of this question a research of literature for both API development challenges and API management challenges is done. The obstacles cover different aspects of API development and management, e.g. design of an API. A further setting taken into account is the development of APIs in ecosystems. To refine the results better for APIs in large organisations, a great portion of the interview phase of this study is dedicated to defining challenges that professional face in practice.

> **RQ2: What are existing incentive mechanisms motivating teams to provide partner APIs?**

Main objective here is to find published scientific work for incentivizing the process of API creation, especially for partner APIs. Incentive mechanisms range from obvious ones, such as monetary, to social, like gamification methods. This study covers both intrinsic and extrinsic incentives. Since there is little literature about incentives in regard to developing partner APIs or APIs in general, the literature review for this question looks into incentives in other areas of software development such as software process improvement, developer ecosystems, open source software and service-oriented architecture.

> **RQ3: What are incentives for providing APIs or API platforms in a large IT organization?**

To answer this question participants from different companies, the majority coming from large IT organizations, are asked to identify incentives that are already incorporated into their work environment or try to justify, if any other incentive mechanisms are applicable to their work. Although the interviewees mention different motivating factors for them, a certain incentive mechanism could not be established based on this qualitative analysis due to the differences in the divisions within the organization.

RQ4: What are recommendations of action for incentivizing developers and architects to provide partner APIs?

A more appropriate way to address the research topic of incentivizing API development and maintenance is creating a list of recommendations of action, when planning an API initiative within an organisation. The list incorporates experiences from various companies as well as from literature. The implications are evaluated by professionals in a follow-up survey.

1.3. Research Approach

A structured literature review has been conducted following the approach proposed in [10]. Goal of the literature review is to identify existing challenges in API development and maintenance and further to find incentives mechanisms for providing partner APIs. Aim is to create a foundation for further research based on interview sessions and identify concepts that are existing and well-studied. Multiple digital libraries such as ScienceDirect and IEEE Xplore, as well as Google Scholar, have been searched.

What are obstacles professionals face, when developing an API in a large organisation? What would help them improve their development and maintenance process? In order to answer those questions, we conducted a series of interviews with 21 professionals with different roles - ranging from software engineers, to architects, to team leads. The interviews included employees of four different companies from diverse businesses. Their input gave us different interesting insights into how they see the API economy developing in their work area and what are the needs for further improvement. An overview of the research approach of this study can be found in Figure 1.1.

In the initial phase, we carried out five conversations with software architects and people from the business to explore the different possibilities for an incentive mechanism and better prepare the questions for the interview phase. This initial phase consists of exploratory interviews that are informal and unstructured. The main focus of the informal talks has been to give initial direction as to how professional could be incentivized in practical setting and what could be leading strategies that are applicable to the company culture of our main industry partner. A list of the roles and the industry, in which the experts are occupied, can be found in Table 1.2.

Next, a series of semi-structured interviews with a majority of experts from our main industry partner and some professionals from other companies has been conducted. In contrast to the exploratory phase, here a questionnaire has been prepared in advance, such that the discussions follow a certain structure and cover main points that are part of the scope of this thesis. Thus, the future analysis towards exploring recurring pattern is possible. The question catalogue has been constructed based on the literature review done beforehand and the insights from the exploratory phase. The questions have been discussed with other professionals to ensure their openness provides enough freedom that interview partners from different industries can answer and still be analyzed holistically. A list of the roles, their professional experience in years and the industry in which they are occupied can be found in Table 1.1.

We interviewed professionals from four different companies – two large IT organisations, a financial company and a mobility provider. The majority of the participants, 24 to be exact, are employees of an international large organisation, that is the main industry partner of this study.

The analysis of the interviews is based on the Grounded Theory methodology [11]. Following this approach a systematic and traceable way of analysis of the conversations is ensured [12].

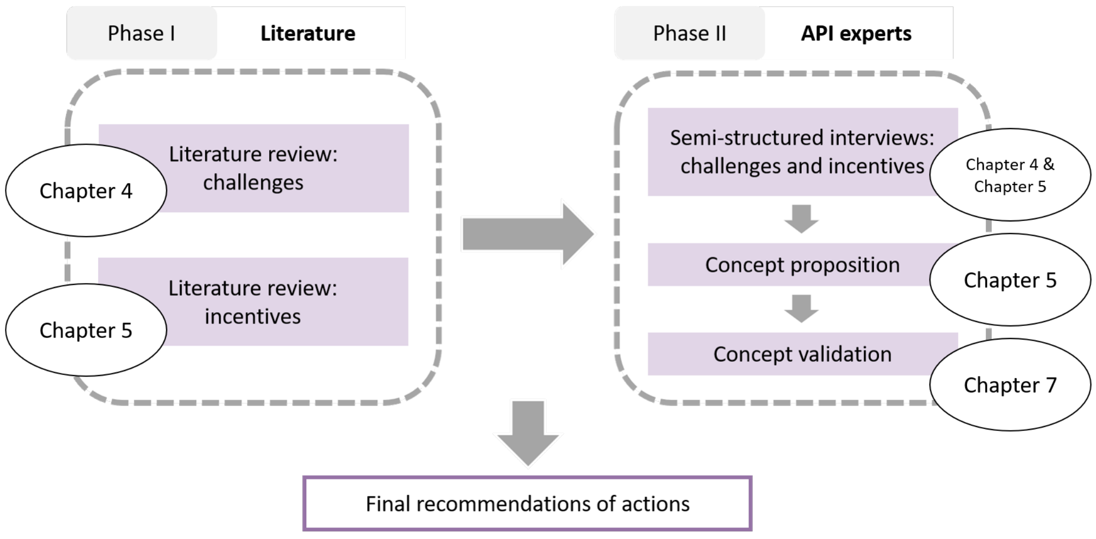
Phase I
Literature
Literature review: challenges
Chapter 4
Literature review: incentives
Chapter 5
Phase II
API experts
Semi-structured interviews: challenges and incentives
Chapter 4 & Chapter 5
Concept proposition
Chapter 5
Concept validation
Chapter 7
Final recommendations of actions

Figure 1.1.: Overview of the research approach.

Alias	Role	Experience	Company
A1	Enterprise Architect	14 years	Banking
A2	Software Architect	6 years	Large tech organisation
A3	Software Architect	4 years	Large tech organisation
A4	Software Architect	5 years	Large tech organisation
A5	Software Architect	>5 years	Large tech organisation
A6	Software Architect	2 years	Large tech organisation
A7	Software Architect	1 year	Large tech organisation
A8	Principal Key Expert	13 year	Large tech organisation
A9	Senior Principle Engineer	9 year	Large tech organisation
A10	Software Architect	14 years	Large tech organisation
D1	Data Scientist	3.5 years	Large tech organisation
D2	Developer	8 years	Large tech organisation
D3	Lead Customer Engineer	1 year	Large IT company
D4	Senior Software engineer	5 years	Large tech organisation
D5	Software engineer	1.3 years	Large tech organisation
D6	Software engineer	3 years	Large tech organisation
D7	Software Engineer	2 years	Large tech organisation
D8	Software Engineer	4 years	Large tech organisation
P1	Product Owner	1 year	Large tech organisation
P2	Team Lead	10 months	Large tech organisation
UM1	Director Software Engineering	4 years	Mobility

Table 1.1.: IDs, roles, years of experience at the given role and company of the interview partners from the semi-structured interviews.

Alias	Role	Company
A12	Senior Software Architect	Large tech organisation
A13	Engineer	Large tech organisation
A14	Principal Key Expert	Large tech organisation
UM2	Head of Digital Service Management	Large tech organisation
UM3	Head of Global Process Ownership	Large tech organisation

Table 1.2.: IDs, roles and companies of the interview partners from the exploratory unstructured interviews.

1.4. Thesis Outline

In this section a brief introduction to the purpose of each chapter is presented.

- **Chapter 1** outlines the motivation to the research topic, defines the research questions of the study as well as the research approach for answering them.
- **Chapter 2** gives the theoretical foundation required to understand the problem statement in depth and to further introduce some important concept of management and incentivization of teams.
- **Chapter 3** presents related publications in the area of API management and incentive mechanisms in different software development processes since little academic literature is given for incentives towards development and maintenance of partner APIs.
- **Chapter 4** discusses challenges in regards to API management. The chapter first presents challenges based on literature and further describes the ones identified in interviews. The chapter concludes with a comparison of both.
- **Chapter 5** provides information about incentives in literature and in interviews. The incentives that are identified from interviews are separated in the ones that currently apply to the professional experience of interview partners and the ones that they see as possible. The possible motivators are modeled as recommendations of action that an organisation might consider when developing an API strategy for entering an API ecosystem through partner APIs. Some of the projects of the main industry partner are presented as their context is important to reasoning why certain incentives are relevant to developing APIs.
- **Chapter 6** discusses a possible game theory mechanism that could be followed to identify such recommendations in an new setting.
- **Chapter 7** describes the evaluation of the recommendations of action. It presents the extent to which the concept is seen as applicable and some further refinements are proposed.
- **Chapter 8** concludes the study by outlining the limitations of this thesis. It summarizes the results and gives direction to future work.

2. Foundations

The goal of this chapter is to present the theoretical foundations for the later work of this thesis. It lays out various terms, concepts and methodologies based on publications relevant for the context of the study. First, the different types of APIs are defined and aspects of the API economy and the API life cycle important to the topic are discussed. Then, architecture concepts that make the creation of APIs easier and some best practices are presented. The chapter concludes with organisational topics in the development process - gamification, game theory and change management.

2.1. API Economy

Before giving a definition of API economy, the ones of API and API management are described. The definition of application programmable interfaces (APIs) is vital for the understanding of the work presented in this study. When talking about APIs often the intention is set to web APIs [4]. APIs give access to software components and therefore are seen as digital assets [8]. APIs can become a starting point for business regardless, if they are invoked inside an organisation or from external partners [4]. The fact that they enable the sharing of data and applications based on standards, makes them a "new form of business model innovation", because they provide functionality for the creating of new offerings [1]. Therefore, in the context of creating business opportunities, APIs are defined as interfaces that are documented, consistent and reliable and establish partnerships between the provider and customer side [13]. The specific type of API, e.g. REST or event-driven, does not influence this definition.

To be able to unlock the capabilities, that APIs can offer, proper API management should be in place. API management includes the activities around the design and implementation of the interface, its release and use. Further, some literature sources include the management of an API ecosystem [9] or the management of developer and business communities [8].

Looking at APIs as enablers, results in placing them in digital ecosystems. When an organisation starts creating or consuming APIs, it forms new partnerships and creates new ecosystems [8]. Thus, the API economy can be defined as the "the commercial exchange of business functions and capabilities using APIs" [1]. The interaction between partners, whether companies, individuals or open source communities, fosters innovation and changes the way business is done in the direction of digitalization [4].

2.1.1. Types of APIs

Although different scientific publications do not consent to one differentiation and definition of the types of APIs, most distinguish between private, partner and public APIs [14]:

- **Private APIs** pair two processes directly. Those APIs in most cases provide access to data. Private APIs can also provide integration components.
- **Partner APIs** represent business relationships.
- **Public APIs**, also often referred to as open APIs, are publicly available and there is no limitation in accessing them.

Other sources [13] group private and partner APIs as one and clearly distinguish them from public APIs, because they see them as much more valuable and as the real driver for innovation opposed to open APIs. Perhaps the most essential differentiation to be made is to identify who is the end-user of the interface and defining its value chain [13]. The API value chain transforms existing business assets into value for the consumers of the endpoint. It is vital to know what is the business value of those assets and how their owner will benefit from them. Just as important is the exposure of the business value in a way such that it can be adopted from the developers, that will consume the API. The consumption comes indirectly from the solutions build on top of the interface. Those solutions are the ones used by the end-users. Taking all stakeholders in this value chain in consideration helps for the creation of a well-designed and successful API. A visualization of the chain is presented in Figure 2.1.

The business value, exposed by an API, can be a further criteria for their type separation [8]. The book [8] presents five different types of APIs in an increasing order based on the visibility of direct revenue they generate. At the very beginning, as with other publications, are APIs that expose information assets. Their pay-off depends on the applications on top. Then come the interfaces that provide access to business processes, resulting in their optimization. Until here those APIs could be generalized as private. The API value chain continues with business-to-customer and business-to-business solutions. Here the value of the APIs is much more quantifiable. The chain ends with APIs for monetization, where the API is a product with a monetization strategy at hand.

From the different definitions of APIs, we see that opening internal systems to others first questions who the consumers of the business assets will be – other parties of an organisation or business partners in the face of third companies.

2.1.2. Drivers for API development

In this section some business aspects as to why organizations aim at API development are presented. One reason is that they easily provide a shift of focus towards the customer experience. Further, they create an ecosystem in which partners easily leverage the value of business assets of others. Another reason is that they significantly reduce the time to market.

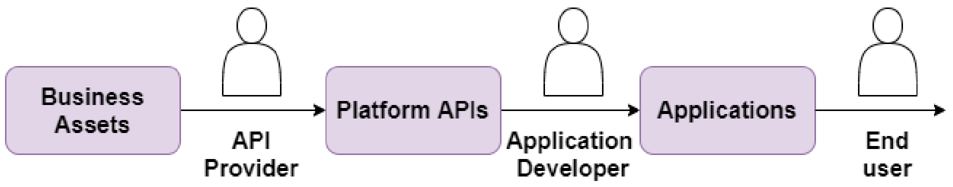

Figure 2.1.: API value chain adopted from [13] and extend to the context of an ecosystem as presented in [15].

[1] Those main advantages that APIs provide fit well to the design of private and partner APIs. Benefits of private APIs include the enablement of fast and scalable development, the simplification of access to business assets. They help businesses create value fast, whereas public APIs mainly contribute to raising the awareness of their benefits. The most important advantage of open APIs is that they represent the digital products of a company. [13]

2.1.3. API Platforms

Many organisations today strategically aim at the adoption of cloud solutions to become customer-centric and achieve digital transformation [8]. Reaching this goal often involves an API platform. Established platforms like Kong [16], offer connection of services to allow developers use the advantages of the newest design pattern. The most important benefits of adopting an API platform for an enterprise lie in the secure management, the scalability and throttling, cashing and monitoring. API platforms assist companies to meet customer demands in a consistent manner. [17]

This study investigates different aspects of API platforms and API gateways. They have their challenges, but are still seen as an important enabler for a proper API management within a company and as vital for creating some visibility and transparency.

2.2. API Life Cycle Management

To be able to have a better overview of the challenges in regard to the processes around API management, first the API life cycle is to be briefly discussed.

The API life cycle in Figure 2.2 displayed below is two-fold. The producer and the consumer cycle are captured – both having their own challenges. The consumer cycle is important to us because we focus on APIs that are to be offered to a partner. It encapsulates the activities of the API maintainer and the different steps represent the further development of the team's strategy. Next, those steps, which are most important to this study, are reviewed.

The whole life cycle starts with the *Strategy* step, where the market to address is specified and the goals are set. This step is a focal point not only when it comes to obstacles around API development and maintenance, but also to the incentives discussed in a later chapter.

The *Design* phase, together with the *Mock* and *Test* phase are seen as challenging steps in literature. It is helpful that they are broken down into to 3 separate steps since in practice these phases could take a longer period of time and are composed of continuous communication between different stakeholders.

Management is as important in the scope of the study as the creation of an API, since goal is to research both development and maintenance. This activity encompasses versioning, deprecation, and retirement.

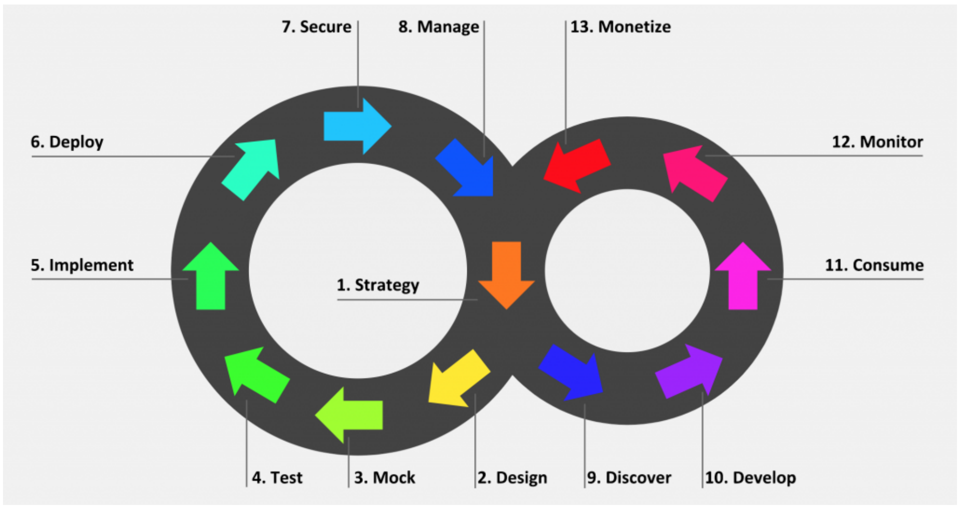

Figure 2.2.: API Life Cycle Management: a detailed view of the API life cycle. The cycle is displayed as two-fold, thus represents both the producer and the consumer related activities. [18]

2.3. Profitable Architecture

In this section software engineering concepts that are important for a good API design and also facilitate the development of APIs are presented. Here, principles such as loose coupling and architecture recommendations are discussed.

One important design practice is aiming for loose coupling in a system [19]. Coupling captures the number of dependencies between modules of a system. When it is loose, the dependencies are few. In contract, tight coupling refers to many dependencies, which might sometimes even be unknown. Service-oriented architecture, for example, should be aiming at loose coupling, so that there are a few and well-known dependencies between the consumers and the providers. [20] This principle is advantageous for the establishment of partner APIs as well.

The good practice of loose coupling becomes easily overseen, because an API is usually build for a specific use case, resulting in an interface with many dependencies with the underlying system [19]. A possible outcome is to have an infrastructure that is "not portable, scalable, reliable, or secure" [19]. In respect to partner APIs, this should be avoided. A tightly coupled architecture hinders the introduction of an API platform, where partners can easily access different APIs. The reasons for the importance of having such a platform or an API registry is also discusses elsewhere [21]. The paper [21] talks about good design practices of enterprise architecture and informs about the advantages of having an enterprise API registry.

The various types of APIs require different types of architecture design. For example, web APIs profit from RESTful services and from having a service layer in the architecture [22].

2.4. REST API Guidelines

Throughout this study often the importance of creating APIs of high quality is discussed. In order to have some common understanding as to what requirements need to be fulfilled in order for an API to be of good quality, this section presents some key best practices for REST APIs. There are various existing guidelines for the creation of REST APIs and an overview as well as a comparison can be found in [23]. The focus lies in REST APIs since they are widely spread and often referred to in the conducted interviews to this study. Next, a few major points for good development practices for REST APIs coming from Zalando's guidelines [24] are discussed:

- **API-first approach**: This design practice is shortly defined in the guidelines as "Microservices development begins with API definition outside the code and ideally involves ample peer-review feedback to achieve high-quality APIs." [24] By first working on the specification of an API, a better level of abstraction can be reached resulting in better design. It also reassures the customer needs are met.

- **Standardization**: Consistency with other teams' APIs and global architecture and obtaining a common look and feel is important especially to external users of company digital assets.

- **API Documentation**: A key factor as to how the API is going to be used is the API documentation. A good API documentation should give information about the implemented methods and endpoints, examples, authorization information and any limits [25]. Zalando's guidelines require a specification with Open API.

- **Meta information**: The API documentation should hold information about the title, version, description and contact.

- **Security**: It is required to use standards, such as authorization via OAuth or signatures.

- **Versioning**: Versioning of an API is a must for the proper maintenance. There are different recommendations as to how to version correctly and the ones of Zalando are compared to others published guidelines in detail in [23].
- **Deprecation process**: The guidelines broadly define a few necessary steps that have to be followed, when an API is deprecated.

2.5. Gamification

Gamification is defined as the use of game design elements in non-game context with the goal to increase engagement and to motivate [12], [26], [27]. Gamification is gaining popularity in the digital economy since its application is transforming the business operations [28].

While there are many publications on how to engage users, in particular developers, to consume APIs, little has been found about the motivation of API creation and more specifically partner APIs. The motivation in software development is however an important factor, directly reflecting in the quality of a software product. The approach has been studied for software process improvement [12].

2.6. Game Theory

Game theory is a generic language for the analysis of multi-agent systems [29]. It finds its application in economics, politics and other social situations, where agents have different preferences and goals [30]. Since the act of providing and consuming APIs includes social and business aspect beyond the technical requirements for their creation, game theory is applicable to the API economy as well. Different agents vary from the development team to the different ecosystem players. Game theory has been previously applied in various software development projects [31]. Existing approaches from game theory that have been studied in economics could be applied to API development so that, first, the goals of all people involved in the process are identified and, second, the process is optimised so that organisations can maximise their profit.

2.6.1. Mechanism Design

Mechanism design is a special part of game theory that concentrates on "social decisions and their effects on outcomes" [32]. Here, the social structure of an organisation is investigated so that "individual incentives of participants can be transformed into the organizational wide desired goals" [32]. Thus, a manager or designer tries to create a mechanism design for the social structure of an organisation. The rules that are embedded in a mechanism design can incentivize individuals to behave in a certain way towards reaching the goals of an organisation [32].

2.6.2. Stackelberg Game

Stackelberg game [33] is a strategic game in economics, that is non-cooperative and includes two types of players: leaders and followers. The game proceeds in the way that the leader announces some kind of information to the followers. They, on the other side, rationalize their actions based on this decision making [34]. The Stackelberg game approach is presented in Chapter 6 for the setting of API development.

2.7. Change Management

Change management in the context of organisations is defined as a continuous process of renewing the organisation's direction, structure and capabilities so that it adapts to the need of both internal and external clients [35]. Change management is important for API management since a successful API strategy drives a development team to become a product-based team. Two main issues in change management have been identified – the rapid pace of change that organisations have to deal with in order to stay in business and the fact that change affects all organisations, regardless if the change is coming from within an organisation or from the outside [36]. Gill et al. [37] argues about the importance of effective leadership in order for change to be well introduced and preserved. Different API management and development roles have to be involved in the change management process, because each one of them has its own responsibilities in an API program [9].

3. Related Work

API Management and API economy: The books "APIs: A strategy guide" [13], "Continuous API management" [9] and "Enterprise API Management" [8] give a detailed understanding of APIs as a technology, talk about their potential in the digital economy and cover various aspects of API management. All three sources are valuable to this thesis since they carefully explain the different enterprise roles involved in the API management. Further, they give many insights about looking at APIs as products. They talk about the architecture designs, implementation patterns and organisation models

API Design: Murphy et al. [38] provides a detailed study on challenges and best practices in regards to API design. The paper gathers experience from various industries and professionals and gives implications of how existing challenges for creating usable APIs could be tackled and informs on best practices from developers' experience. Main points in the paper are designing towards a good API user experience, designing to most important use cases rather than including all, enabling users to easily begin consuming the API and quick discoverability.

API management challenges in ecosystems: In [39] four main challenges in API management ecosystems are defined from case studies. An API is investigated from its technical side and as a business enabler. Although the challenges are defined for internal ecosystems, the three ecosystems where the case studies were conducted are platforms with both internal and external partners. Hammouda et al. [15] further studies the challenges of API design in ecosystems. He stresses on the importance of the continuous development of APIs that are part of a platform. The paper describes the API value chain in terms of an ecosystem.

Software development incentives: Incentives are studies widely in software areas different from API management. Baddoo et al. [40] informs on motivators of software process improvement. The motivators in that area are insightful for API development and maintenance as well. The paper stresses on the importance of the motivation of people involved in the process in contrast to technical factors, that are easy to comprehend and manage. The author bases his findings on classic motivation theory and divides them across different groups: developers, managers and senior management.

Haruvy et al. [41] studies the two traditional models of innovation in the setting of open source development. The author argues that both the private investment model and the collective action model cannot justify the success of open source software. The given incentives could be related to API development in the aspects of monetary incentivization, self-development and receiving gratitude.

4. Challenges

This chapter describes challenges in API development and maintenance and aims to answer the question of defining obstacles for providing partner APIs in internal solution. Section 4.1 intends to summarize existing challenges discussed in literature. Section 4.2 presents a qualitative study of challenges, that have been identified in the semi-structured interviews. Both chapters are structured by the pillars of the API life cycle. Finally, the findings from literature and interviews are compared.

4.1. Challenges in API Development and Maintenance - Literature Review

This section presents the results of the literature review. An overview of the challenges can be found in Table 4.1.

4.1.1. Strategy

Business value

Different strategy applies to the different types of APIs. One can distinguish between private, partner and public APIs. However, in some literature, partner APIs are addressed together with private APIs. When it comes to API strategy, it is important to define the API value chain [13]. The success of the API is determined by the correct understanding of the business asset, the API provider, the developers (the ones who consume the API) and the applications, meaning the end users of the business assets. The value chain can be further extended by incorporating an API ecosystem [15], which could be a part of an enterprise's API strategy. As described in [13] a private API can support a partner relationship, when it is used for building applications of value to the partner. In case the partnership becomes more automated, the book categorized the API as a public one. Although partner API as a term is not used, the strategy regarding the business value chain is still relevant to it. It is challenging for actors to correctly define the value chain and to continuously deliver value [15].

Quality

In an enterprise setting, the development of new APIs is in the majority of cases part of a project and done by requirement. However, certain conditions have to be met to be able to develop a good API (see section 2.4 for a definition of widely applied standards for REST API). When a platform is also involved, two further challenges arise. First, the API has to meet the requirements of the ecosystem and, second, a assessment for that has to be established. These challenges should be addressed in the strategy phase prior to definition and development of an API. [15]

Ownership

While the goals set for an API are the focus of the strategy step, something else that is not to be underestimated are the stakeholders involved in the process and more specifically the development team. There are different strategies if the development team and the maintenance team are the same or two separate groups of people. Finding a person, who will be responsible for the API design is discussed in [38]. The question of responsibility around APIs is something that will also be further discussed in the interview results, as it has been expressed as a concern by multiple professionals from different branches.

4.1.2. Design and Implementation

The design phase of API management is perceived as the one with the most decision-making. The design decisions made at this step of the process impact the whole life cycle. For example, an event-based API influences the implementation, deployment, monitoring and documentation. [9].

Design

One of the main challenges of the design of an API is to determine the use cases that are most valuable for a good design [38]. The paper suggests that reviews could be very valuable for developers, which implies that it is currently missing. "Early feedback from users on an API's design and future use cases will result in a better design, but was reported to be challenging to obtain." [38]

For the implementation of an API, guidelines are usually followed. Either from the open source community or specific to the organisation at question, guidelines can be confusing and not well fitting to the specific use case, for which the API is designed. This leaves developers with open questions. [38]. Guidance become even more important, if the API is intended as part of a platform, because certain standardization has to be created. [15].

A challenge is to decide who is doing the API design – is it coming from the project manager/project owner or from the developers [38]. An example given in the paper is when the API is about e.g. financial data. Then it might be better to have the design from the service manager. On the other hand, when it is an API that is for database engineers, then the design should come from the development team. In both cases there should be involvement from both sides.

Developers' education

Further, the education of the development team is to be taken into account. Good APIs are designed by experienced developers and the better the quality, the higher the probability of re-usability of the API since it will have better API usability and better documentation. This challenge has been discussed in [38] and [39]. The challenge is summarized by [39] in the sense that regardless of the experience of the developer, the design and evolution, the whole API life cycle, calls for a different mind set and therefore, continuous education is advisable. API developers learn best through practice and based on feedback gathered from peer reviews. The lack of specialization of software engineers in API design is discussed as well, leading to an API design not oriented to the API user. More training materials are needed. [38]

4.1.3. Deployment

A good interface is only recognised as such if its documentation is well written. Both cater for a good user experience resulting into more strategic value [9]. Here, some challenges that arise for providing a good documentation to the consumers are summarized.

Documentation generation

There has been a concert as to the automatic generation of software development kits and documentation of APIs. Although there are existing solutions to automate the process, their fine-tuning is still missing, creating output that does not fulfill requirements. [38].

API Portal

A further challenge is the lack of a portal designed for the developers, who consume the APIs, where they can find documentation, SDKs and the APIs themselves. [42]

4.1.4. Security

In literature, APIs in different industries are studied. One problem that arises with the development of technologies is the adequate security of APIs. In internet of things (IoT) devices, developers are challenged to find new ways of securing communication, since well-established and studied ways for authentication and authorization cannot be applied to a device-to-server communication. [42]

4.1.5. Management

Management challenges in regard to API development and maintenance found in literature are mainly concerning the evolution of an API in the context of a platform or ecosystem.

Versioning

Based on the progression of a platform, APIs have to evolve accordingly [15]. An interface has to change guided by the data gathered from usage. "For the API to evolve, a regular, systematic, and quantifiable assessment has to be planned." [15] How advanced the API to interact with the platform is a decision based on the user base of the API and could be changing as more feedback of consumption is gathered.

In the book "Continuous API Management" [9] the author specifies three main challenges for the API life cycle management. The first is the strategy of API governance of a central architecture board. It is difficult to create a strategy that could be applied across many projects and secure a high level of control over the API quality. Second, the scaling of the team, that is managing the APIs, has to be according to the growth of APIs in the organisation. Last, it can be difficult to establish a set of rules for the standardization of the interfaces, that does not constraint the projects to decide on details based on the specifics of the projects. These challenges are all with the presumption that an organisation aims at a central API strategy.

A further challenge, in the context of working with partners, is the innovation speed of evolution. The difficulties come from the different speed with which the partners can migrate. Their motivation to migrate is bounded by their business goals. [39]

Deprecation

The expected behaviour of an API is that it is created, versioned, deprecated and retired. These steps are the foundation of its life cycle. While in literature, design and maintenance are well studied, the deprecation process is often neglected.

In ecosystems, the deprecation of an API needs to follow a clear strategy. This becomes an obstacle for development teams, since it is expected that the API will be maintained and kept stable [39].

4.1.6. Discovery

Discoverability of APIs

In the discovery phase, it is important to promote an API so that it reaches consumers and they start using it. A problem that has been previously encountered is the creation of a documentation, which makes the use of the API easy. It is important that the documentation consists of examples and is easy to work with, so that consumers are not put off by it [38]. A detailed study of what makes a documentation useful for the API consumer and aspects how to overcome obstacles related to API documentation shows that documentation with intent and scenarios are vital for a good use [43]. The API documentation should be done with good developer experience in mind, so that they can, first, indicate the functionality of the API is useful to them and, second, start using it easily.

Discoverability of platforms

The challenge discussed above becomes even more important, if the quality and usability of the API are a premise for a consumer to be involved with a platform [15]. An API design that does not satisfy the user needs and has to be changed hinders the platform to become a provider for stable solutions.

4.1.7. Monitoring

After the release of an API, its providers find it difficult to obtain customer feedback due to the lack of tools [38]. Even if there is a channel, where information about the usage of the interface is at hand, the interpretation might be troublesome. The challenge to identify the usage of an API by the partners in order to optimise it is also discussed elsewhere [39]. The management of dependencies can vary in difficulty based on what kind of platform it is talked about: centralized or shipping application type.

4.2. Qualitative Analysis of API Development Challenges

In this section the challenges identified in the conducted semi-structured interviews are presented. The obstacles are from various projects that include developing and maintaining APIs. As described in 2.1 the API life cycle consists of multiple phases and the challenges are sorted according to those. An overview is displayed in Table 4.2.

Since in the interview phase specialists from different companies and with different positions have participated, the described challenges below are further distinguished by those criteria as well. This is needed since companies are in different stages of their strategy development when it comes to the API economy and market and, therefore, the difficulties, that the employees of the firms have, differ.

4.2.1. Strategy Challenges

In the strategy phase of the API life cycle, goals are set. Defining targets includes various challenges mainly because all parties have to decide on a clear strategy.

Requirements satisfaction

In the case of a project that is in an early phase of definition, interview partners [D2, A2, A3] have identified as challenging to cover different requirements of different use cases. For example, creating an API gateway means diverse requirements have to be fulfilled by it, ensuring all use cases. Further, problematic is ensuring the availability of data at all time for all use cases. Having multiple use cases also challenges architects to decide on a overall strategy for the gateway, especially when scenarios come in after initial strategy planning.

> "Every app can have different requirements and these requirements need to be fulfilled by the API gateway. We have to ensure these use cases. The gateway should be able to communicate to all the services at all time." [D2]

Another challenge expressed by multiple informants is estimating the need of an API before it creation [A8, A10, UM1]. The need of the API has to be well-justified. This holds for

Source	Strategy	Design	Mock	Test	Implement	Deploy	Secure	Manage	Discover	Develop	Monitor
Web API Management Meets the Internet of Things [42]						No portal to publish details of the APIs, documentation, SDKs	Standard protocols not applicable for all devices		Developer portals		Monitoring usage control
Continuous API Design for Software Ecosystems [15]	Define API value chain; Continuous value delivery; Meet ecosystem requirement; assessment of API for ecosystem	Designing and implementing APIs that satisfy the ecosystemability need;						Continuous API evolution		API quality and usability	
API designers in the field: Design practices and challenges for creating usable APIs [38]	Lack of dedicated API designers; Lack of training resources; finding responsible for API design	Discern valuable use cases; No previous knowledge of API design; Guidelines not good enough to define the whole development process; Lack of consistency in the design; Good abstraction	Getting peer reviews	Obtain early feedback from customers	No obvious design best practices	Automatic generation of SDKs and documentation - not tuneable enough		Change in the API is hard due to existing dependent software	Writing proper documentation to increase discoverability	Usability problems	Gather feedback and interpret it
Continuous API Management [9]								Lack of centralized strategy for API governance; Scaling; Standardization of interfaces			
API management challenges in ecosystems [39]	Lack of developers' education; Lack of continuous education							Different innovation speed; Management of dependencies; Lack of deprecation process			

Table 4.1.: Challenges of providing partner APIs based on main sources of literature review. The challenges are sectioned by the phases of the API life cycle.

both directions: having a use case and creating an API out of it or actively searching for a service or functionality, where the development of an API would be a beneficial investment. Both are found challenging since they demand communication between different stakeholders.

Challenging topic that the developers themselves see in the strategy phase of API development, a key aspect that is recurring in the interviews, is the communication. For example, communicating the API requirements with different stakeholders such as product owners and architects can be difficult even when doing a proof of concept (PoC).

Developer community

A strategy obstacle for the establishment of a gateway is also considering that opposed to other platforms, an API gateway within an organisation has much less developers contributing to it [A7]. To reach a critical mass on the platform becomes troublesome since fewer people need to create a lot of value fast.

When working with a monolith system, whose architecture is to be rebuild into a microservice architecture, the creation of APIs is a task that may come as new for the developers. Thus, lack of previous experience and lack of guidelines to follow is identified as a challenge that some teams have to overcome on their own.

Ecosystem needs

From an organisational point of view, challenges are connected to how to become a ecosystem partner without overexposing valuable assets. [UM1, P1, D3] Where there are many competitors on the market, a discussion arises which APIs are to be opened and which not. And further, how to protect them. It is challenging to move away from an isolated position that one company can produce a full stack that is a product and think more of an ecosystem approach. Interview partners see that it is hard to give up on the intellectual property but is the only way an ecosystem could exist. But profiting of foreign assets means that one's products shift from being isolated products, where the amount of value is limited just by that one company.

> "So, if you book directly on a hotel website, you get some benefit. And this is basically the war in the travel industry. I mean, you need the aggregator [websites]. On the other hand, you want to fight them a little bit. And this is why we always have the discussions, if we should open APIs or not." [UM1]

The last challenge, in regard to strategy, is how to measure the business success of an API – based on consumption, based on increase in revenue or customer gain. [D3]

4.2.2. Definition Challenges

Standardization

Defining standards for the APIs with the responsible teams or the service owner is seen as problematic [A1, A2, A10, UM1, P1]. This is important for both provider and consumers. Providing teams have to follow certain standards for the users to have a coherent experience when consuming different APIs.

> "Because, in my view, the challenge with API design is when every project does if for themselves. Then it is quite likely that every project will also implement a good API, but if you have them side by side, they have nothing in common. As I said, that's why, in my view, the challenge is that the APIs are also to some extent uniformly designed, ideally based on a domain data model." [A1]

Part of the strategy is also to establish good design practices. It is difficult to ensure extensibility of the interface, to require developers to put extra effort into designing re-usable APIs. Discussions arise because although an API that is to be used for multiple use cases is desired, at the time of design usually one use case is at hand. Thus, many assumptions have to be made and additional effort is invested without a promise for new use cases. [A1, A3, A10]

> "If I don't have more number of people or applications that will use this, the extent of reuse of this API gets limited. This API would essentially be only used for one or two applications that exactly need these values here. If I need something else, somebody has to go back and update the connector or build a new connector." [A3]

However, some [A9] argue that it is better to have the same methods that are in multiple APIs, but to create role specific interfaces for the different user groups or for the different access levels of one service.

API Design

The majority of interview partners have identified some main challenges in the definition phase of the API life cycle, independently of their role. On one hand, it is challenging for the development team to utilize the requirements for an API as good as possible. [D1, D4, D5, A6, A8] On the other, defining the API to be understandable by user and providing a good documentation is also troublesome. Both are next discussed in detail. [A2, A9, A10, D8]

Understanding the specification requirements is challenging because the specification of APIs is often not detailed enough or not clear enough. When requirements are broad, developers have to interpret those requirements differently and take ownership of the decisions made. Often the development team has to convince stakeholders, that the presented solution is suitable in order to reach some end results. [D4] Developers express that product owners usually have some requirements that change over time. This introduces a risk for the team to reach deadlines and to produce a good software. It depends on the team – if they would stand for the specification or they would be agreeing to changes. [D5] This is understandable because part of designing an API is making decisions on the proper design of an API, the granularity of the design of an API, how much to expose from an internal system, at what level to expose. Further, an open question is if one should create a single API, that exposes a lot of the data, or more fine-grained APIs, that expose very specific aspects of the data. [A8]

Defining the API to be usable and understandable is further a challenge that developers face. A good API, that is also to be used by partner, have to ensure high level of usability. [A10]

> "One challenge is always the other end point, in this case, the consuming part of the API, that we establish a common understanding of the API. In this case, we had a two-way communication, so, he can send messages to me and I can send messages to him. So, there was a lot of back and forth and redesigning the API until both parties had a common understanding." [D1]

The API should be built in a way that different attributes are understandable. [A2] An API has to be usable, that means, especially when it is given to the outside, it is very important that the API is expressive, so that the intention is visible in its signature. When the method names are hard to understand or there are very fine-grained methods, the consumer is left on his own to interpret the API specification and, therefore, there is a lot that can be done wrong. Another important aspect for the design of an API is not to let users create their user scenarios by calling multiple methods in a sequence, but to cover as many scenarios as possible in the definition of the API and have them ready to use. The functionality of the API has to come from real and concrete workflows. [A9]

Often a documentation that is not detailed enough leaves some open questions, such as where can the API work, what kind of data can it work with, what does it support and what does it not. And all those should be documented along with some examples. [D8]

Although there are tools available for creating documentation, usually there is a gap between the documentation and the implementation of the API. [A6]

A lot of the challenges concern mainly REST APIs since they are widely adopted. However, one further challenge is to recognise when explicitly defined interfaces, where there is a clear definition of which method to call to obtain certain output from a service, have to be replaced by interfaces, which incorporate event storming. It is challenging to orient the development to domain driven design. [A9]

4.2.3. Deployment

The deployment process also has its obstacles. Depending on the tools used for deployment, developers do not always see the process as straightforward, for example deploying multiple instances of the same service in the context of microservice-based architecture.

Cloud services have become widely used, however, there is a lack of proper authentication for developers to deploy to the cloud, leaving interfaces stored on local machines, where no visibility could be created. [D5]

4.2.4. Security

Security is an essential step of the API life cycle. Many developers and architects have expressed different concerns in regard to security – how to strictly follow the security constraints of their organisation, how to quickly ensure security.

Security standards

Further, it is challenging to come to an agreement on how to best provide security. Both internally and externally, services need to be exposed securely. [D3] On an organisational level, questions such as what security protocols, identity and authentication is to be used, what is a platform that is to be established to securely publish APIs internally and externally for consumption, are usually still open. Technical teams are missing those decisions to publish their APIs and get their services to be used. Instead of having a lengthy process, teams require a way to publish things quickly. However, company policies that are in place prevent them from publishing APIs directly to external users. For each interface, details like the Accredited Service Provider (ASP) level of the data and communication though firewall need to be defined [D1]. There is a lack of authentication and encryption following organisational guidelines, that would potentially reduce operational costs for e.g. setting an authentication process. One architect [A6] sees the need of a centralized way for rate limits, billing and authentication. The issue is that a lot of APIs are available, but then one has to have a special account, which is to be obtained from the API owner. When a company has an authentication solution, which covers all cybersecurity requirements, it should not be the case that publishing API is such a challenge. Further, an identified challenge is that solutions that work within the intranet of an organisation, have to be adjusted for the internet.

> "When we created [the platform], we thought it's a platform, you install it and you're done. But we spent pretty much three years for talking to the security officer, talking to data privacy officer, to export control people." [A6]

Partner relations

Even if the security from the provider side is good, there are obstacles that arise, when working with partners. One of the industries, included in the interviews, has already multiple partnerships at place. However, when the security of their interfaces has to be improved with some functionality, partners lack the involvement for support. The main reason is the additional costs from partner side, regardless if the partner provides sophisticated IT solutions as well. [UM1] Implementing the authentication from partner side can also be a long and troublesome process. [A2] Further, it is hard to ensure that an API has only the functionality wished by the consuming part and no further. It is challenging to ensure that a service is providing only access to a certain data base. [P1]

In a gateway, multiple data sources can be accessed by different consumers. Interview partners, especially developers [D2, D7], have said they encountered the problem to expose data to multiple clients. They needed to restrict it in a way to ensure the client who is requesting data is authorized to get the data from gateway. Further, deploying a project in a secure way has also been defined as a challenge.

4.2.5. Management

The management phase of an API life cycle includes versioning, deprecation and retirement of an API.

Versioning

Introducing changes to an interface has been a widely discussed topic from many interview partners.

Introducing breaking changes is often described as a reoccurring problem, when an API has to be adjusted. [A4, A8, A9, A10, P2, UM1]. Even when an API is created, such that it satisfies the initial use case for its creation, when something on customer's side changes, adjustments in the API usually need to be introduced as well. When changes become breaking changes, there is a challenge for the development team. The developer team would prefer to reduce development and maintenance costs by introducing changes. That is the optimal solution for it to do the refactoring. However, this way the customer satisfaction goes down. The other way around is to build something additional, so that the consumers do not have to deal with breaking changes, and they receive the new code separately, which gives them more time to integrate it into their system. Which solution to choose, in the context of partner APIs, becomes a challenge since the goal is to ensure long term stability for the partner. And further, if not all consumers of an interface are known, then communicating errors becomes difficult. It is important to ensure support to the users, otherwise if the partner has many problems using the interface, it is likely that he would not want to further use the software of the company, since APIs are an important digital asset.

> "Especially when you have a lot of partners out there and a lot of partners with legacy systems as well. For them it's not just okay, upgrade to a new API contract version. For them it takes years to adopt. And this is also very challenging there." [UM1]

Therefore, introducing changes has to have solid reasons and the API has to stay stable, if possible, when extending it, adding new features, refactoring and introducing breaking changes.

Backward compatibility is another challenge developers and architects have to tackle [A4, A5, A8, A9, D7]. Again, it would be demanding for maintenance to change a method in a widely used interface, because everyone is affected. Providing more specific interfaces, means more maintenance work. However, if a change of a method that is only in one, for example, role-specific interface, only the users that have the specific role are affected. Teams need to ensure that evolution is possible in an interface, for example, by using extension methods, extension interfaces and other patterns or language properties and only then consumers who are really interested in the change are affected.

Some architects do not see how one framework for API management would be feasible for a whole enterprise. They thinks that maybe a central gateway for each business unit is doable, but different business unit have different speed and this would be a problem for the projects, which work in shorter time periods and, thus, with different cycles, because the requirements would slow them down. Their differences in strategies requires different approaches. [A4]

Deprecation

Deprecation is part of the API life cycle, that interview partners rarely have touched upon, since most APIs they own are internal ones. However, one informant [A10], that is part of a platform with different consumers, discussed that deprecation is challenging since partner APIs are expected to run always. But this is not executable and so it is to be decided case by case what deprecation strategy will be assigned for the API at question.

4.2.6. Discovery

How to make an API visible within the organisation and where to publish it has been identified as a challenge [A6, A13, A12, A14, UM3]. Big leaders in the API economy have dedicated platforms for APIs, such as Google and GitHub. In the context of the industry partner even existing standardized interfaces currently have no place where they could be discovered.

> "It's always a struggle to find who can provide the data or the API. There are some APIs, but not documented, and they are very hard to use." [D1]

Further, challenges for exposing an API outside of an organisation come in terms of export controlling customs, open source software clearings. This is where development teams would rather leave the responsibility to the business side. [A3]

4.2.7. Consumption

One challenge is assuring data availability. [D2, D4, D7, P1] In the case of an API gateway, this becomes a challenge since it is enough for one underlying service to be unavailable so that the whole transaction has to be rolled back. It is hard to make that layer more robust. In a microservice-based architecture developers need to ensure availability of the data and have to deal with concurrency.

A central gateway has to filter out parameters that are not of interest to the customer and to also provide different names for parameters coming from different connectors.

Developers also struggle to achieve a stateless behavior it terms of a call of a REST API.

4.2.8. Monitoring

Developers find monitoring of microservices difficult, when talking about a workflow rather than the separate services. For this they lack the appropriate tool. Although some open source solutions are available, a lot of additional work from the developer's side is needed.

4.2.9. Monetization

Some interview partners have expressed that they see a lack of monetization strategy [A3, A6, UM3]. For a monetization strategy to come in question, there should be indicators that there is a customer base, however, this usually takes time. Without a critical mass, there are lengthy discussions about what the base for monetizing should be. Although APIs are recognised as important, the lack of business plans of how to monetize such APIs postulates creating a front-end and maybe a mobile application for it. This is seen as one roadblock, which prevents from exposing an API to the outside world.

4.3. Comparison of Literature and Interview Challenges

When comparing the challenges from literature review and semi-structured interviews carried out in the scope of this study, it is important to keep in mind that most of the challenges in the interviews are regarding internal APIs, whereas the literature review prioritized challenges for providing partner APIs.

Even with this main difference, the results in regard to the strategy phase of API management are quite similar. An example is the continuous value delivery in an ecosystem – our interview partners fear that an organisation is a community not large enough to provide new features in a timely fashion. In literature, the challenge is more focused on how not to lose control of valuable assets in the platform and keep them evolving according to the evolution of an ecosystem.

Both sources define a need for better education of developers. A similarity is that both suggest that a development team is capable of technically providing an API. It is rather the knowledge of good API design that is intended.

Something that is missing in the literature review, but is discussed in the interview sessions, is the need of an API. This difference comes mainly from the fact that the literature sources are gather with the intend to created end-points to be exposed to third parties, while in practice, teams want to spare additional costs for the creation of an API and thus always question the need of such. This should not be seen as a setback since it provides a higher level of quality.

Defined by:	**Architects**	**Developers**	**Architects; Developers**	**Architects; Developers; Upper Management**	**Product Owners; Upper Management**
Strategy	Fewer developers on internal platform	Communicating requirements with stakeholders; how to measure success; Lack of guidelines, lack of experience	Cover different use cases	Estimate need of API	Overexposing valuable assets; Change management: ecosystem player
Design	Extencibility; Design with respect to reusability; Data protection; Gap between documentation and implementation	Specification too broad; Requirements change over time; Writing good documentation	Usability	Defining standards	
Deploy		No authentication for cloud deployment; Deploying multiple instances			
Secure	Internet and intranet differences; no centralized way of authentication	Provide data access to correct users; platform with security protocols	follow organisation restrictions; Secure quickly		Partners not collaborating for ensuring security
Manage	No deprecation strategy		Backwards compatibility	Introducing breaking changes; communicating errors to consumers	Partners adopt slow to changes
Discover	OSS clearings			Visibility: publish API and documentation	
Monitor		Missing tools for monitoring			
Monetization				No monetization strategy	

Table 4.2.: Challenges of providing partner APIs based on semi-structured interviews. The challenges are sectioned by the phases of the API life cycle.

The design in both sources is mainly discussed in terms of usability. Both stress on the fact that the design should be with intention to consumers. While experts from our interviews worry more about the proper communication, when presenting and reviewing an API with different stakeholders, literature sources report that there is a lack of a steady review process at all.

In contrast to literature, interviewed professionals expressed their concern to create a documentation that is understandable. Some literature sources strategically inform on a challenge one step further, namely writing documentation that makes the API discoverable. Both talk about the lack of sophisticated enough software to automatically create documentation out of implementation, that allows creators to fine-tune the output.

The challenge of deploying a service in practice differ from literature. Developers included in the interview phase face problems accessing cloud platforms for deployment, while literature resources inform on the lack of a dedicated platform at all.

Security is studied in two very different aspects in literature and in the semi-structured interviews. Experts have to overcome obstacles with respect to the organisational requirements, that they have to follow, to secure their services. They inform on the lack of a standardized authentication process within their organisation or on the slow speed with which the security is provided. On the other hand, literature sources cover more technical aspects of security.

Regarding management of APIs, both sources discuss similar aspects that are rather technical. It is important that the evolution of services is carried out well to support the customers. While literature resource talk of changes as a necessary step for APIs to evolve and continue being used, our interviewees comment on changes as a direct requirement of users.

While in literature, the need of a clear deprecation strategy is reported, interview partners say that it depends more on the specific case and it is not applicable to have a standard process in that regard.

The discovery phase of the API life cycle is as vital for creating revenue of the API's business value as is the API itself. Both sources report of a missing dedicated developer portal, where APIs could be made visible.

Both comments of interview partners and results from the literature review show that there is a lack of tooling for monitoring. In literature, further the strategy of how to correctly interpret monitoring data is discussed. Interviews do not have such information with exception to one industry partner, that states this is an important point for the future development of teams and products.

None of the literature resources talk about monetization directly. One of them reports on the importance of defining a proper API value chain, which could indirectly be associated to monetization, but is seen more as a strategy aspect. In practice, missing monetization plans hinder the proper establishment of partnerships based on a service.

5. Analysis of Incentives for API Development and Maintenance

This chapter presents an analysis of the motivation for creating APIs, existing incentives and a concept proposition for incentivizing developers and architects to provide APIs. The results of the literature review that aims to identify existing incentive mechanisms motivating teams to develop partner APIs are presented first in Section 5.1. For the better understanding of the incentives that come from industry partners, a few projects that part of the interview participants have been contributing, are introduced in Section 5.2. Next, current motivating factors based on the semi-structured interviews conducted in the scope of this study are summarized in Section 5.3. Those are compared to literature findings in Section **??** Last, recommendations of action for incentivizing developers and architects to provide partner APIs is suggested in Section 5.4.

5.1. Literature Review

A systematic literature review is performed in order to research available sources for the topic area of incentivizing API development and API management. To date no scientific publications are found that fit the research criteria. Therefore, other areas of software engineering, such as software process improvement, developer ecosystems and others have been studied. Incentives from those, that would be applicable to the development and maintenance of APIs, are presented in this section. An overview of the incentives can be found in Table 5.1.

5.1.1. Gamification

To incentivize developers to develop specific tasks, often gamification is applied since it enables us to define mechanisms to encourage motivation. In one publication [12] the creation process of a gamification network for software process improvement (SPI) is described. Afridi et al. [44] discusses an approach to effectively reward developers, contributing to open source software (OSS). However, the lack of gamification frameworks for providing APIs of any kind and the feedback we have received in the exploratory interviews, has shifted the focus of incentive mechanisms for providing partner APIs away from gamification and towards other strategies to maximize the developers' commitment.

5.1.2. Project Management Related Incentives

In the context of SPI, motivating factors have been studies according to classic motivation theory [40]. The author lists the most common motivators for three types of positions – developers, managers and senior management. The paper divides influencing factors in extrinsic and intrinsic. Extrinsic factors are maintaining factors and as such they do not provide satisfaction with work. Intrinsic factors are motivational factors and they attain satisfaction. The leading motivators from the publication [40] are visible success for all three groups and additionally bottom-up initiatives as well as top-down commitment for the developers. Visible success is defined as evidence of benefits [40], which is also applicable to partner APIs, since some existing and successful partnership should be at hand in the organisation. Creating visible success by having a few high-profile products attracts other developers [45].

An incentive for developers to create APIs is to enable them to generate revenue. One way is to provide them with a platform, where awareness is created and marketing strategies are available. This gives them a financial motivation to contribute to a platform. [45] Further, it is important to create a feeling of contribution to a successful platform. One way to do so is to create developer programs based on the crowd-source approach [45]. In the context of open source software (OSS), successful stories also are identified as incentivizing to the contributors. Because there are many projects that indicate success, programmers are motivated to work on OSS. [46] The paper also distinguished the motivation behind bigger and smaller projects, which are altruism and the act of gift-giving accordingly. While creating successful stories about APIs within an organisation is achievable, the latter two motivators are intrinsic and do not fit so well to API development.

Service-oriented architecture (SOA) and APIs are comparable design practices [9]. Identified incentives for developers in SOA are the creation of innovation and the shift of mindset towards IT value [47]. The reason behind is that those architecture designs enable automation of processes and speed up innovation.

5.1.3. Recognition

Empowerment is an important incentive for project managers and is defined as "practices within the SPI program that empower staff to take decisions on changing processes" [40]. This could be applied to API development as well, because the project managers are the ones who best know what the business value of a potential API would be and should be empowered to prioritize the creation of an end-point along with other tasks of a project. Project managers from the publication [40] also speak about process ownership as a motivating factor.

Some incentives from open source development could be considered for API enterprise development. For example, undermining the power of large software houses in OSS can be understand as ownership culture in API management where employees want their company to be a competitor on the market. [41]

The fact that open source software initiatives and API development can be used as tools for providing innovation in a system makes some incentives of OSS applicable to API creation [41]. The paper defines ego gratification and self-fulfillment as private investment model motivators. They apply to corporate development as long as there is a platform for a developer to receive gratitude for his work – either from peer reviews or from management.

Even though small gifts are not the best approach to incentivize developers, some more advanced techniques of showing gratitude like creating challenges that are rewarded with badges show some positive outcome [45].

5.1.4. Technical

Since providing partner APIs as a strategic asset is in a sense additional work to a project that is not vital to it, it would be understandable that having sufficient time and resources allocated to API development would be motivating to teams. This motivator has been well-identified across all three roles studied in [40].

There are various platforms on the market designed for APIs. However, for a developer portal to thrive, there is the need of a constant evolution and activity. Technical motivators to commit to using such a platform are, for example, simplicity of integrating software. Developers should be incentivized by engaging them with "self-service access to developer materials", developer programs, sample code and technical support [45].

When researching incentives to provide better security measures for software development, Halderman et al. [48] stresses on the importance of transparency. An incentive, which applies to API development is the transparency of the processes which is achieved by introducing best practices.

5.1.5. Self-development

Beyond having online resource for self-improvement or webinars and online conferences [49] it is even more valued to have trainings on site, developer gatherings and to foster the exchange of ideas by creating a community. This is also a good way for starting new collaborations. [45] Evidence for motivators like exchanging knowledge and creating new forms of cooperation are also found in topics like open source development [41].

5.1.6. Monetary

The paper [41] defines the incentives for contribution to open source project under two categories – the private investment model and social considerations. Private incentives include monetary motivators, such as job prospects, promotions and bonuses. Developers contribute

Source	Project management	Recognition	Technical	Self-development	Monetary
Motivators of Software Process Improvement: An analysis of practitioners' views [40]	Visible success; meeting targets; bottom-up initiatives and top-down commitment	Recognition by senior management; empowerment; process ownership	Resources		
Building a developer ecosystem: What vendors do to attract you to their platforms [45]	Help with marketing; creating awareness; existing successful stories; crowd-source approach;	Small gifts; challenges with prizes; badges;	Easy software integration into platform; sample code; quality tech support;	Developer programs available on premise; free downloads for self-development; community and collaboration	
Incentives for Developers' Contributions and Product Performance Metrics in Open Source Development: An Empirical Exploration [41]		Receiving gratitude - from peers or management; ownership culture;		Exchanging knowledge; creating new forms of cooperation	Job prospects, promotions, salary increases
If Open Source Code Is a Public Good, Why Does Private Provision Work(Or Does It)? [46]	Success stories; promise of successful work				
To strengthen security, change developers' incentives [48]			Guidelines		
The future of enterprise applications [47]	Innovation; shift mindset towards IT value				

Table 5.1.: Incentives found in literature based on various software processes.

to a project to signal their ability to potential employers. In the setting of an organisation, where a developer is already employed, his incentive would be to show potential and interest for further development in the firm and would expect a salary increase or promotion in his already existing position. The motivation of career advancement has been found in different papers about incentives in OSS and have been summarized by [50].

5.2. Use Cases

Many of the challenges and incentives, identified by the interview partners coming from the main industry partner, have based their examples of three main projects. In contrast, informants from other companies have talked about various examples from their work experience.

API Gateway

The first project is about the establishment of an API gateway within a company. For the

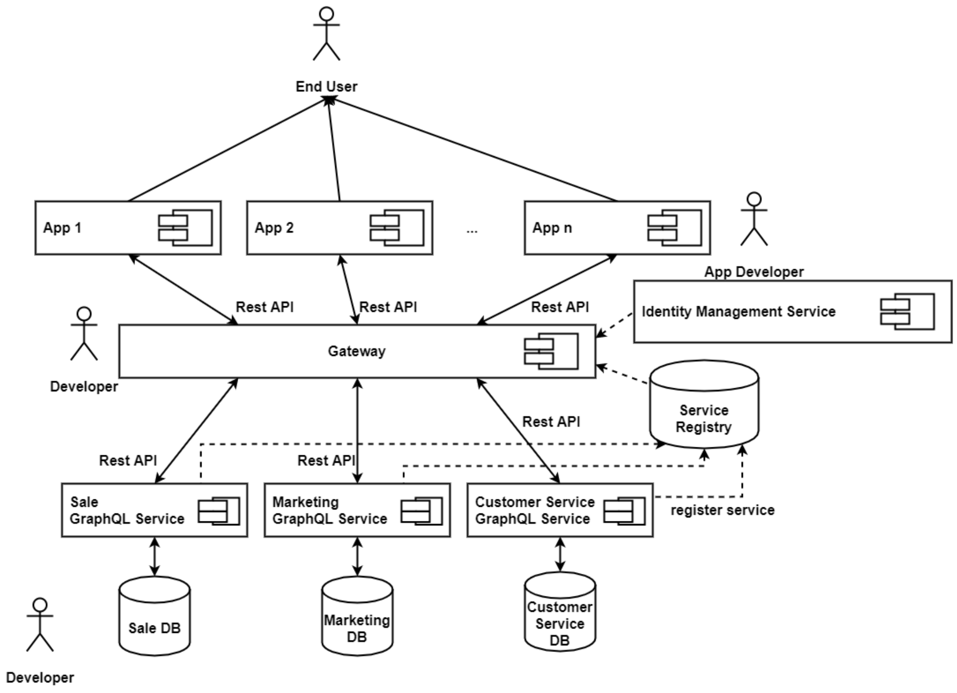

Figure 5.1.: Customer Relationship Management Architecture

purposes of this paper, a toy example for customer relationship management architecture is displayed in Figure 5.1. The three data bases in the architecture are taken from [51].

Important for the scope of this study are the places in the architecture, where APIs are developed. In the bottom part of the architecture different heterogeneous systems are displayed. These are raw data sources. If some application wants to access this data, they should know how to connect to the source, to know the schema of the data sources. The gateway aims to represent that particular data in the underlying source as a conceptual model and expose this to the end applications. This is done across all data sources of interest. Further, the access to the data is encapsulated in the architecture. That is how the application can access the data though an end-point. Next, the gateway provides an abstraction, where applications directly talk to a semantic model instead of having to query the data sources individually and then do the aggregation within the application logic.

Therefore, there are two main places, where APIs play an important role. One is to provide data from the data sources to the gateway and second is to provide data to the applications.

API Developer Portal

The second project is already in operation, even though its initial design (see Figure 5.2) has

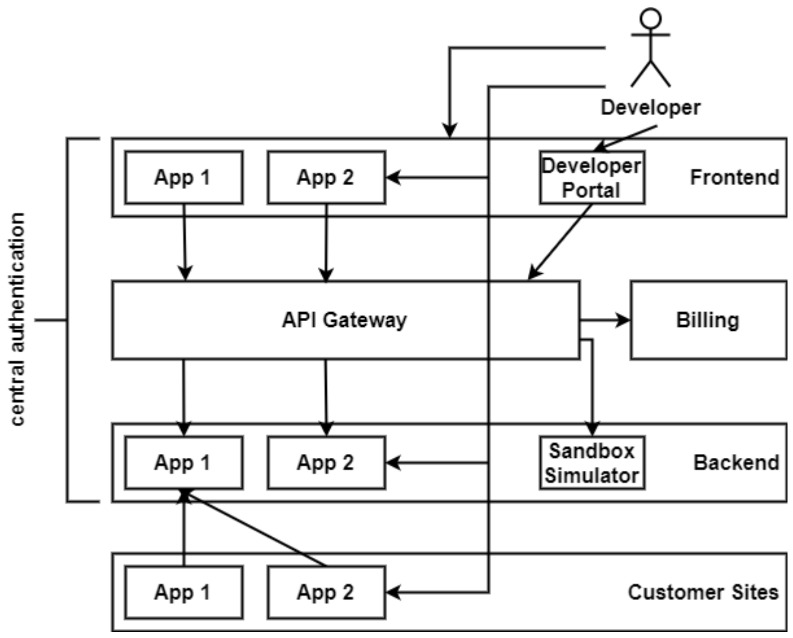

Figure 5.2.: Overview of an API platform

not taken place and is currently constricted to a static web page. The project aims to form a centralized API platform within the organisation. Key features of the architecture are to have a central authentication service, an API gateway and a billing system [52]. The project also foresees a developer portal for application development.

A important feature of the project is the conduction of peer reviews for all APIs listed on the platform. Thus, a higher level of quality is reassured.

API Platform

The last project is an established platform, which is oriented towards providing APIs to partners. The topics included in the platform are all related to the internet of things (IoT). The offerings hold ready-to-use APIs and services. The platform is cloud based and provides extensive support for developers and also offers a partner program for developers with incentives, such as trainings, marketing and revenue generation.

5.3. Current Motivation

In this section some motivating factors that did not find application in the concept of recommendations 5.4 are discussed. They have been mentioned in the interview round and either have to do with the advantages of using APIs or the intrinsic motivation of the professionals.

5.3.1. API Properties

Monolithic Architecture to Microservice-based Architecture

In total 5 architects, 3 developers and one person from upper management mentions the benefits of APIs as a technology as a motivating factor for them [A1, A2, A3, A4, A6, D4, D5, D6, UM3]. It is a common pattern for large organisations to have multiple legacy systems, that have monolithic architecture. The complexity of the existing systems is challenging for a migration to take place. To lower the complexity, APIs are often used. This is where a lot of interview partner see the potential of APIs and also have gathered their professional experience.

Developers know that REST APIs are easy to implement and that they are a suitable technology to support this migration. The good community support and documentation available gives a certain level of comfort, when reducing the complexity. With the modularity of a microservice-based architecture the system becomes easier for maintenance because of the reduction of the number of dependencies. Introduction of changes to the system also becomes less complicated since not the complete application has to be newly deployed. One should instead deploy the particular microservice, where the change occurs.

While introducing breaking changes, when providing an API is a difficulty, defined by many, it is easy to find errors when using APIs and, therefore, one can easily keep the service in check. The overall traceability of APIs makes them widely used. APIs have the advantage that one can easily monitor them. Calls can be traced, and the API usage can be traced. From there the load can be detected. An example is the tracing solution of Amazon Web Services (AWS) called AWS x-ray. For cloud-based applications such a tool gives transparency about usage of application. It is not always the case that more important APIs are the most expensive, so costs can be saved based on the monitoring. Participants see the possibility that monitoring can be strategically used for identifying future development of the project or the team.

Overall, the granularity that APIs bring as a technical advantage results in less effort in development and testing and intentional communication with product owners of specific modules or domains rather than the whole systems. Also, the granularity ensures no dependence on a single technology and gives the opportunity to choose from a range of technologies.

API are, on one hand, an easy starting point for a project, when developing software, on the other, they are simple to communicate with other stakeholders within the project. After an agreement on them, more sophisticated modules of the project can be discussed, such as a front-end application.

5.3.2. Introducing Features

The introduction of new features is seen as motivating mainly by developers [D3, D5, D6, UM1].

Development teams recognise the need for introducing new features. Simple, but valuable smaller software, in the form of APIs, gives teams the possibility to enter an ecosystem.

An example is digitalization architectures. APIs are how teams are to offer their services in the future. With all the digital offerings and digital solutions that are coming through and also all across business use cases that are available today, APIs are the basic building block that are needed to be offered in order for such solutions to come into play. With more demands of such cross-domain use cases coming in and more demands of web technologies for digital offerings, it becomes extremely important to have access to such building blocks.

Teams stay aware of what already exists, identify a need for further development and study how to proactively engage customers with new features based on trends in the industry before there is a requirement for such. There is a constant interplay between innovation and demand and successful teams usually work closely together to correctly prioritize the two.

5.3.3. Transparency

Reaching a high level of transparency has been identified as a main motivation for the involved parties by many. [A4, A6, A8, A12, A13, P2, UM2]

The best incentive mechanism for the developers would be for them to know what is the customer value that they bring by developing an API. Currently this is not the case. The product owner is the one selling the product and developers do not have the chance to understand where their input is going [A12].

Seeing the API usage data as a key performance indication could further be motivating. It shows how well the developed product is being adopted, how much it is being consumed. [D3]

> "And I think that's also kind of where the incentive comes for developers. So they actually see: hey, what they're doing is going to be used and can be really easily used." [A6]

Transparency fosters the intrinsic motivation of developers and is very important for an organisation, if it wants to move forward in an innovative way. The understanding of the purpose of a project and how it is being used is how new potential for further development can be identified.

Transparency is one of the main incentives as to why large organisations target to establish an API platform other than all technical advantages of such platforms. Based on the consumption of an API from a platform, the extent to which the API is integrated in other projects and the way user interact with it, insights, as to how well the professionals behind the API are working, can be obtained. Experts, who are part of this study, find this as an important incentive for their future work, even when there is no explicit customer feedback. To see that their time and energy investment pays off is just as important.

Through APIs the altruistic motivation can be enhanced as well. Being both provider and consumer at the same time, being able to exchange expertise available via APIs within the organisation.

5.3.4. Alignment of Business and IT

One of the companies included in this study is a multinational technology company, that is currently one of the leading enterprises in the API economy, and also offers its solutions and experience to others. Its experience regarding alignment of business and IT towards a cohesive API strategy is presented below. Some other interview partners also have commented on the alignment of business and IT [A6, UM1].

It is an important step to have business and IT aligned and to build the same strategy for them to follow. The consultancy tries to always position both the business value side or the organizational aspect and the technical side. From a business perspective it is important to consider what is being published and how it is being consumed – internally and externally. They provide API-coaches in the team who work with the business side of customers on the API strategy and the importance of APIs in today's world. Thus, through evangelism API initiatives are created within the business and the organisation. The technical side is being addressed on topics such as security of publishing APIs both for internal and external use, authentication, identity, platform where to publish. The end goal is bringing together business and tech in the organization and having a holistic view of APIs as important strategic assets that can be build out. [D3]

5.4. Concept

5.4.1. Data Sharing

In today's digital economy, companies increasingly work together with ecosystem partners. Data has become a valuable resource and the more resources are available, the better and more precise insights are to be found. Research communities have been developing strategies to incentivize openness within the community [53]. Although Morey et al. [53] discusses the importance of data sharing in science, the approach that is described could find its application in the API economy as well.

> "An aspect that's very important is moving away from an isolated position that one company can produce the full stack that is the product and thinking more of an ecosystem approach. Because APIs offer the integration with your partners, with customers, with even competitors that increase your value proposition immensely." [D3]

There are different platforms that have recognised the importance of data sharing within an enterprise [54], and also between multiple companies [55]. The described platform [55] lists several advantages of having a single data platform for sharing data. Some of the main benefits are reaching a high level of security, subscription verification, prevention of resource duplication and analytics for consumption.

A previous study [52] within the context of the industry partner of this paper has drawn five main conclusions that developers of APIs want to have "unauthenticated access to resources (documentation, SDK and code samples)", "active and responsive community (large user/consumer base)", "local independent testing", "login with an existing account" and "delivery of applications (e.g. hosting, updates)". The first of the above is the main goal of data sharing, while the latter are preconditions for such a data exchange to take place.

Taking into account the interviews conducted for this study, multiple professionals have expressed that they see having an active data exchange through APIs within the organisation would incentivize them to also actively provide their data, that is of value. In particular, two architects, three developers and one person from upper management commented on the topic [A6, A12, D1, D3, D4, UM1].

> "...altruistic motivation. But also with GitLab we are both provider and consumer at the same time. And we create the whole API topic because we want to consume it. If you have so many really talented people at the company, which are skilled in their specific topic and making their expertise available via API so that we can consume it. That's really awesome." [A6]

Another quote from a data scientist also providing APIs:

> "What needs to be done [...] is to have a meta-data repository for the data sources around the company. This is step number one. We have some data scattered

around and we need a central place where the data is described - where it resides, who is the data owner, some sample queries." [D1]

Based on all of the above, a possible solution for nurturing the creation of APIs for opening existing internal systems is creating a central place for data exchange. A must in the context of APIs is having the documentation. A good documentation is required for high acceptance of an API. Developers, in the sense of consumers of an API, expect an easily accessible, well-written documentation with various code examples. Having code examples in the language of programming enables consumers to incorporate an API fast. Otherwise, the use of an API could only be based on a demand. A data sharing platform should also include the way of getting access to the interface since multiple experts have expressed their concern that obtaining access is based purely on personal networks.

Recommendation: A central place for data sharing will encourage the professionals to open their systems. A platform should the least hold the data description, data owner and a way to access the data.

5.4.2. Speed

Time to market is a central point when working with partners. Here, both the speed of the developing process and the speed of adopting an API by the customers is discussed. The topic of speed has been mentioned by four architects, one software engineer from a leading company in the API economy, one product owner and two people from management [A2, A3, A6, A7, D3, P1, UM1, UM3].

Interview partners from different enterprises [A7, D3] have talked about the importance of external access by self-service where any customer, who wants to integrate existing solutions, can immediately go to an API portal and be in the position to start testing APIs. A next step would be to register and get access to APIs. All those steps should be supported in a self-service way, so that less time is spent on communication and better customer engagement is achieved. One architect [A6] pointed out that this is would be beneficial within the organisation, where he is working.

The majority of the participants in the interview round have gathered their experience in API management and development from internal systems, where a trusted network is at hand. An important aspect of opening those internal systems to partners is exposing functionality in a secure way. This easily becomes a blocking point. Becoming an API provider goes hand in hand with publishing and managing APIs in a secure way. Thus, it is important that an organisation willing to enter the API economy and to work with partners, enables development teams to publish securely. There are existing cloud solutions that provide such functionality or an in-house solution could be established. Either way, a platform, where the APIs can be exposed becomes necessary for ensuring a high speed to market.

"Technical teams want to publish their APIs. And they want to update and move

> fast and publish things quickly. On the other hand, you have the security and the IT and other stakeholders who have the opposite view. That things have to first be totally secured. Of course, which is important, no question, but there are company policies in place that just prevent teams from publishing APIs directly to external users." [D3]

Further, both customers and developers should be able to easily understand the given APIs to start working on their own user stories. [P1] Having some interfaces available to the customer, already enables him to solve certain problems. It solves the connectivity problem and the authentication problem. It solves how one can represent a scheme. One can directly extend such interfaces and does not have to figure out the whole process from scratch. Therefore, it is much faster to build a custom API by extending some generic solution, where a template or source code is available. [A3] This approach provides a way for departments inside an organization to publish APIs quickly, effectively, automatically, without any manual intervention. [D3]

Recommendation: Speed is determining in today's digital economy. Therefore, both API providers and consumers should be able to interact fast. Developers should be able to publish APIs securely and to provide authentication in a simple manner. Customers should be able to quickly understand the APIs and with the help of self-service engage with the provided solutions.

5.4.3. Communication

Creating a usable API, therefore, an API that can be easily adopted by partners, depends on having a clear vision together with stakeholders. Since partner APIs are limited to specific partners in contrast to open APIs, where the consumers are unknown, communication with potential or existing users can be established. The experience of our informants is mainly based on the creation of internal API, however, some implications relevant to partner API could be made. Four developers, four architects and two other professionals have expressed their opinion about communication [A2, A3, A5, A6, D1, D5, D6, D7, P1, UM1]. Our exploratory study identified transparency as one of the main incentives when it comes to API development. Active communication can be separated in communication between the development team and the different stakeholders in the design phase of the API and communication between the API providers and consumers in the case of establishing a partnership.

Each role, that is involved in the process, has different responsibilities and thus has pointed out different important aspects. Developers find it important to closely work with product owners and iteratively discuss the design of an API [D5, D6, D7].

As an example, when re-building a monolith system into a microservice-based system, developers decide on an approach by doing a proof of concept (PoC) and presenting and discussing it to architects and product owners within the project. By working closely with the domain expertise of product owners and the technical knowledge of architects, developers feel that they narrow down possible development decisions and test the best ones. By doing so, an agreement on how to proceed with the project is reached.

Efficient communication with product owners is especially important to developers not only to stay on the right track, but also do deliver value on time. Both developers and architects have recognised APIs as small building block that are easy to create. Therefore, an API is the first thing that a developer team can actually show to the customer. They give a base for both sides to start a discussion and proceed with more advanced and time-consuming development such as a front-end application, which involves user experience and user interfaces and can become challenging to make it right from the first attempt. However, since product owners have some requirements that change over time, it is a task of the developers team to decide to that extent they would be working with the initial specification and where they would agree to adjust their work in order to implement new ideas.

> "REST APIs are also not that easy, but at least it's sometimes, in the initial phases, it is easy to modify it, if you have implemented something wrong." [D5]

Both architects and product owners, who are informant is the semi-structured interviews, acknowledge the importance of communicating with the customers or partners as well:

> "Because we try to really be customer focused. I don't really mean like customer be the one who defines our product but also like who consumes our API. We had the feeling that if you already start coding around and then, you know, just see the interface finished, the Swagger file and the artefact, you might end up with an API, which is quite easy to implement, but it's not really what the customer wants."[A6]

It is challenging to establish a common understanding of an API and explain the API itself to the users. This usually ends up in long discussions with partners. [A2] In order to create a usable API some architects implement peer reviews, when developing APIs.

Communicating with partners also helps receive support in their business domain as to what kind of conceptual model is to be exposed and what semantics are to be build on top. Therefore, architects see a trend, where successful teams work agile and closely with experts with domain knowledge. [A5]

Organisational roles see the importance of communication for teams as an incentive, since often the demand for an API comes from the business. That is why one has to redefine what is a team and to include the business and the engineers into one team and that is how making business becomes the incentive for the engineers as well. [UM1, P1]

Recommendation: Transparency is a key factor for the motivation of development teams. An efficient communication between stakeholders and developers to get feedback on time should be planned. Working together with the business or with people with understanding of the domain, especially in the design phase, is important for creating a good API design taking re-usability and the business value of the API in account.

5.4.4. Top Management Initiatives

While in literature, stories of existing success are identified as an incentive for developers [46], [40], interviewees that have won awards for innovation for their API initiatives within their organisations [A1, A6], say that is it a great starting point, but a lot of support is needed further in time in order to establish a change in strategy for a bigger community. In both companies the award is given for projects and the focus has been on opening up internal ecosystems to the customer. The opportunity to monetize existing data and services as well as the role of APIs as enabler has been recognised.

> "People now are really looking for reusable asset or building blocks, call them however you like. I think the awareness is necessarily there in higher management, I would say also our product managers. People learned a lot more of how to monetize APIs. So suddenly we have a monetary and project related incentive ongoing." [A6]

Management involvement is especially important since there is a shift in the way of work and the way the organisation is perceived. Top management initiatives have been described as important by 5 architects, 3 developers and two further experts [A1, A3, A6, A12, A14, D1, D3, D8, P1, UM1]. To enter the digital economy, new structures should be established, focusing on the people and engaging them to reshape their old way of work by increasing their involvement in their topics. For this to happen, top down initiatives should take place, because it is sufficient when a few people do not support the topic of API management for it to have enormous issues to move forward. [P1] Therefore, it is important to at least create a virtual space, where successful stories and valuable experience is shared and where the API topic is highlighted as with the awards, discussed in the beginning of this section. This way an API initiative in the organization is started, bringing together tech, business, IT, security. This is an existing strategy of leading companies in the field [D3]. They have seen that the importance of APIs is well known in the technical teams, but they do not feel understood from business or from the organizational side.

Success within the API economy is based on working on both business value and organisational aspect. For the development teams it is vital to identify the assets that are published, their consumption patterns by internal and external partners. On the other hand, the business side should discuss on the API strategy, if APIs are included in the business strategy at all and where the focus lies. By doing so an API initiative is created in a company and it is being stressed on the importance of APIs. [D3]

> "There are several ways of spreading the awareness. One could be for example we have groups in our communication platform, which are there. So you could have

> an API community within there, where the best practices of APIs are documented there or some information is provided there. That would be a good way to create an API community, at least which increases the awareness." [D8]

Companies that have already established active partnerships based on opening their internal systems via partner APIs, talk about future steps, that would not be possible without top management involvement:

> "So basically, that's even a business strategy. Basically, go away from being a car rental company into becoming a provider mobility platform." [UM1]

Recommendation: There is the need of a strategy shift towards API economy on an organisational level for teams to actively engage in the topic.

5.4.5. Project Organisation

For a successful strategy to enter the API ecosystem, a shift in the mindset of development teams is seen as vital by various interview partners. In total, three architects, 6 developers and two people from upper management and a product owner have expressed the importance of a proper project organisation [A1, A2, A3, D1, D3, D4, D5, D6, D7, P1, UM1, UM2]. Teams have to change their strategy towards delivering a product.

It is important that when developing an API, the end-to-end process is thought trough. The main question should be "Where is my product standing?" and that only after identification of the business value, it can be proceeded to architecture and development. The existing functionality of a team and where it is standing in the value chain should be a motivating factor for the creating of an API, that results in an offering of a product. [UM2] This change results in a product-based team: the API developers should be a part of a multi-disciplinary team, that is organised by product. It should be based on the product life cycle and not project phases, working towards continuous delivery. [P1]

Further, change in the way of thinking is desired, namely by working with the unknown rather than presenting a well-known solution. Everyone in the team should take ownership of the delivered value instead of leadership taking over ownership. [P1]. This is a key aspect because when an API is delivered based on existing requirements, the specification can be very broad, resulting in people having to interpret those requirements differently and not wanting to make a decision. [D5]

> "But we don't want to lose speed in what we're developing. So, a lot of times we have to make assumptions and assumptions that this might be relevant, or this might be important. And we start building it. But parallel also crosscheck with the concrete business unit whether this is useful or not." [A3]

Further, multiple developers, that have participated in the interviews, have pointed out that it is vital to think domain driven, when discussing about their API-related projects. [D4, D6, D7] Especially when working with microservices, working more in depth makes it possible to deliver high productivity with a particular module. This helps that particular microservice to evolve more. A well-developed microservice in a specific domain can be then distributed horizontally in different domains, for example, a separate service, which is responsible for authorization or authentication.

Recommendation: The project organisation should centre around the creation of a product. Tech, business, IT, security should work together and move away from the strategy of an isolated player to the idea of an ecosystem player through APIs.

5.4.6. API Gateway Essentials

In the exploratory interview phase of this study, some participants have pointed out that a platform, where an API could be made visible, is crucial for teams to provide partner APIs. In this section the extent to which it is applicable in various projects is discussed. A gateway is seen as advantageous by 7 architects, three developers and two further people, who have participated in the interviews [A1, A2, A3, A4, A6, A7, A14, D2, D3, D6, P1, UM1].

Interviewees from different businesses have commented that for them an API gateway should cover technical aspects, such as authentication and authorization. [A1, A4] On the other hand, domain functionality should be out of the scope of a gateway, resulting in a split of the technical side from business side, thus, having a simpler API gateway. In the context of working with a lot of partners, [UM1] expressed that the way a API gateway is used in their firm is for defining internal quality gates. An API needs to be versioned. If someone encloses something to the public and if they change something in the infrastructure, they need to introduce a version. Another feature is a centralized locking for the APIs, which means monitoring the number of requests coming in, the load of the underlying clusters.

One of the projects, on which some interview partners have been working, includes an API gateway and has the following design:

> "Apart from the connectors themselves and the semantic mapping itself, in future we are also going to provide a monitoring end-point that will let you monitor what you have deployed in terms of its heartbeat, how it is performing, some measurements and KPIs." [A3]

For an API gateway to function and teams to willingly publish their APIs on a gateway, there should already be some working interfaces existing on it. Building an API gateway means to consider the following:

> "As a developer, you will do something if you see that is going to pay off. And so, there's an initial problem of starting up an ecosystem - is where you don't have anything in the ecosystem." [A7]

It is essential to reach a critical mass, before establishing an API gateway within an organisation. [A1] A first step is to have enough existing APIs operating as services and then make the next step towards building an ecosystem. But once this critical mass is reached, the partnership between a provider and consumer moves its focus from actively reaching out to integrate consumers to getting the consumer to utilize the services provided. [UM1]

Different projects call for different adoption of an API gateway. It needs to be considered that not all projects are suitable for such a context. An ecosystem thrives when there is no manual work involved, so one needs to have self-service. Big consulting projects do not fit well, because of the underlying complexity. Their complexity is not a negative feature but rather the quality of such projects and products. [A7] When products are too complicated and represent different systems, it is preferred to have separate platforms for each one of them instead of a centralized one. [UM1] However, an API gateway is meaningful in a microservice-based architecture, where each service has a specific task and consumers cannot have the overview of all the available functionalities. Thus, the level of granularity is a key factor as to when an API gateway is meaningful.

Recommendation: An API gateway contributes to establishing internal quality gates, to provide versioning, to do monitoring, to have authentication and to secure availability. The aspect of security is leading for both architects and developers, so that the authentication is according to the organisation's requirements.

5.4.7. Guidelines

At all organisational levels guidelines and best practices are seen as obligatory. However, what practice will be applied depend on the specific use case. Seven architects, four developers, two people with management roles and one product owner, speak about guidelines in the interviews of this study [A1, A3, A4, A6, A9, A10, A14, D2, D4, D6, D7, P1, UM1, UM3].

The majority of developers [D2, D4, D6, D7], who participated in the semi-structured interviews, already follow some guidelines for developing APIs, but seek the help of architects when issues arise. They find it important to have good community support and documentation available and to be able to get guidance from architects, who worked on different domains and have a broader view of the technology. Their broader knowledge also helps developers identify projects with similar issues, that were already encountered.

Since some technologies, such as RESTful APIs, are more mature and therefore have better documentation and best practices available, newer technologies such as GraphQL require more support until they prove as sufficient to many services by the evaluation of the status of those services in real time. [D7] Another example is the fact that domain-driven design is something that is more and more asked for in order to break a monolith structure, but engineers struggle to think in event-driven design and go back to developing REST APIs, which are sometimes not used at all or are even hindering the project. Thus, guidelines and best practices have to be schooled in a very hands-on way to be really taken into practice. [A9]

From an architects' perspective, guidelines are essential to ensure a certain quality of the APIs. Partner-APIs have to be created at a standard since once they are published, changes require additional communication with consumers. Thus, quality specification and standards are to be followed, e.g. in an API gateway [A4]. Architects, that have tried to establish a platform within their organisation, have also done multiple workshops with global architects of several divisions to define guidelines and systematize the creation of APIs. Guidelines do not need to be very specific, since that would make them inapplicable to a wider community of developers. Having some ground requirements towards the quality of the APIs helps to further estimate, if an API is actually needed. [A6]

A successful way of working is leaving the development to the project team rather than having a central API development team. However, once there is a need of an API, the requirement should be evaluated outside of the team. In order to best implemented a specification in an API, some guidance should be given, reassuring better quality [A1, A10].

Non-technical roles such as product owners [P1] and leads [UM1] have also pointed out that for them it is important to have a centralized architecture board, that ensures a certain level of standardization. Without such a mechanism, that secures a minimal set of rules, an API catalogue would be impractical to customers.

An important remark is that all participants in the interview phase of this study have been chosen so that they have experience with API development. However, guidelines and best practices are vital for teams that want to enter the API economy by opening up their system.

Recommendation: Creating a minimal set of best practices and standards with the help of experienced professionals to be followed within a organization is essential for a company to later provide new and existing APIs to partners.

Re-usability

Although re-usability of APIs is not highly prioritized when creating an API, multiple informants have pointed out that re-usability should be encouraged and that a balance between use case specific and too generic interfaces should be found.

When architects talk about complex system with various applications, they see the possibility of reducing the complexity by creating re-usable APIs. Their experience shows that they have been able to identify cases, where requirements are very similar, e.g. when obtaining data from a system. [A1, A3] Guidance in this case is needed in order to have an API that is not too generic. [A10] A very generic interface without actual consumers will result in extra cost in terms of development and testing.

When asked about what kind of interfaces (connectors), with respect to re-usability, are looked for in an API gateway, the following has been shared:

> "One way we are approaching it is to provide generic connectors which are not bound to application specific terminology. And that way we encapsulate the protocol and we encapsulate provider generic model. The more generic you get, it means that use cases might not be able to use it directly. So they will have to have a semantic connector on top of the generic connector we provide, additionally. For us, one goal is to promote reuse of such connectors, so that everybody does not have to develop from scratch. But when you provide anything generic, you will always have this challenge that it is not tailored to my specific need and you have to do some effort on top of it." [A3]

One architect [A5], involved in a more mature system, which receives requirements for APIs from customers directly, commented that their approach is to always adjust the demand in a way that the API is generalized, so that the requirements could be further applied to other customers of the system as well.

Implication: Re-usable API reduce the complexity of a system and save development and maintenance costs to teams. Therefore, it is advisable to design an API as re-usable without making is too generic for the initial use case.

5.4.8. Development Tools and Environment

Technical challenges, although much easier to overcome in comparison to others, are not to be underestimated. Four developers, three architects and a product owner, who are interview partners, have commented on the topic [A2, A3, A6, D3, D4, D5, D7, P1]. Making tools for API development and maintenance available for the developers is a necessary step to enable teams to publish quickly, securely and in a managed way.

REST APIs are widely used and a lot of tooling is already available. Informants in the semi-structured interviews wish for more support and access to such [D5, A2]. Otherwise, the development process becomes more problematic and complicated, resulting in time loses.

When commenting on future plans of further development of an API platform, architects include aspects in regard to API maintenance:

> "In future we are also going to provide a monitoring end-point that will let you monitor what you have deployed in terms of its heartbeat, how it's performing, some measurements and KPIs." [A3]

Such tools are vital for a company to be prepared to offer its APIs to partners and although licensing costs will arise, the benefits of such tools should be explained, similar to the advantages of the cloud-based services, that are now part of most teams work [P1].

Recommendation: Development environment should be enhanced with necessary tools for teams to easily create and maintain APIs of high quality.

Closing the gap between documentation and implementation

Especially in big systems, where there are a lot of APIs, architects have experienced that it is hard to keep the implementation and documentation at the same level.

Even with existing established solutions, such as Swagger, there have been some minor problems, that should be solved in the future.

> "We wrote the Swagger documentation and used it for the code. So that is kind of the one or the other way. And with both ways, we actually noticed that there was always a small gap when it comes to action documentation versus what is really implemented. It is not something bad or hindering or critical. But it's just something which we struggled with. I would say automation in that area is not that great at the moment." [A6]

Recommendation: There is a need for a tool that enables API developers to automatically create documentation out of implementation or vise versa with very high accuracy.

5.4.9. Monetization Strategy

Monetization is part of the API life cycle and is especially relevant for partner APIs. When asked about their monetization strategy, most participants do not have an answer since their work is in the context of internal APIs. However, having initial monetization strategy at hand means future intention to enter the API economy as an ecosystem player. Two people at leading positions, three architects and one developer have talked about monetization [A1, A3, A6, D3, UM1, UM3].

> „But with our business management, the sales, we are not really there that we can monetize an API. [...] We kind of seen the importance of APIs, but we had no business plans of how to monetize such an API rather than creating a front-end and maybe a mobile application for it. And really leveraged the API's way to scale and iterate quickly rather than actually earning money with it. That was certainly one roadblock, which prevented us from opening up an API to the whole world.“ [A6]

The monetization strategy is not to be left completely to the business but should be collaborative work of different roles involved in the process. Experience from established API platforms suggests approaching the topic by first seeing what the actual usage of an API is. By putting out new APIs and obtaining the data of first users using them, analysis to see where monetization can be effective can be done. A strong monetization strategy at the beginning or in the wrong ways will result in a lot of users put off using the APIs, and thus will increase the barrier to entry. A better approach is to first publish APIs that can used, or any registered developer can use, so that enough data about consumption is gathered. [D3]

Once consumption is evaluated and usage has intentionally grown, tears or base offerings that anyone can use are to be introduced. Afterwards, the monetization strategy should be evolving and be flexible instead of having fixed ratings. Such data-driven model makes the whole team to reach better business decisions in a fast manner and be more flexible to the changes in market. The way APIs are consumed requires an understanding of how reoccurring revenue works. This is a necessary change in the way business is perceived in order to understand how digital revenue models work opposed to license-based models or traditional models.

Seeing APIs as products, which could be priced, means having a multidisciplinary team, that does not separate development from maintenance, bur rather works on all aspects. [P1] The decision of which interface is to be provided to partners should be made by the ones who have the business insights and have identified potential use cases. [A1]

Monetization is key aspect of the way a platform is developed. Mature platforms follow the business strategy to build and offer everything as a service. [A3] Starting an API platform at a company, where APIs are to be provided to partners, calls for potential customers to be identified and also decide on which assets are to be monetized, such as connectors, applications, usage. A decision at what level of granularity will the pricing take place has to made.

The incentive is to be able to offer a visible and transparent plan how maintainers of complex systems can monetize their changes and how they can get payments for the traffic. There is a need for a process of commercial marketing that enables providers to make their APIs visible fast and also to enable consumers to be also to get access to the APIs fast. [UM2]

Recommendation: The monetization of an API is not to be underestimated. This step of the API life cycle calls for the domain expertise of the business to support teams. The business side needs to adjust its monetization strategy to the way APIs are consumed.

6. API Development Process as an Economic Mechanism

As a result of the literature review and interview result, it is visible that APIs in large IT organisations are often created on demand, sometimes to introduce new features based on the expertise of developers and architects involved in the process and rarely to purposefully expose the business value of an internal system for the sake of creating new business possibilities.

Incentives vary from business to business and from product to product. Therefore, it is of interest to know how one can model such mechanisms and apply the approach to a specific case. Since, based on this study, the business value is key factor for the creation of partner APIs, software economics should be considered and game theory as a theoretical framework could be applied for optimal decision making and maximization of productivity.

To date, no specific game theory methods have been found for the development of APIs, however, Yilmaz et al. [56] proposes an economic mechanism design to improve software development process, which fits very well to the scope of this study. The aim of the mechanism is to "find ways to adjust the incentives and disincentives of the software organization to align them with the motivations of the participants in order to maximize the delivered value of a software project" [56].

Identifying the need or the potential interest for an endpoint relies on communication with stakeholders and working with different use cases, thus, it could be considered as a result of a set of social activities. [57] Social aspects, such as knowledge sharing and motivation, make the application of economic modelling useful for designing a game theoretic model for the API life cycle. [58]

What makes the economic mechanism presented in [56] suitable for the social setting of API development is the fact that information is considered of economic value, which in our case is the information given by the API. The development and maintenance can be looked at as a production and distribution of organizational services, which makes it an economic activity based on information exchange economy. Based on those similarities one can model the development process as a Stackelberg game [33], which is an interaction model, consisting of leaders and followers in terms of game theory. In this particular model the leader chooses a strategy at first and followers then act second as to maximize the outcome. APIs are part of software products, which are usually developed by teams consisting of team leads and developers. This means that in the economic model the team lead is the leader and the developers are the followers, who have the main objective to maximize the business value of the project by creating an API. The team leader could also be a software architect in some cases. The Stackelberg game is formed by a manager, who in this particular scenario should be the product owner.

Next, the steps to reach equilibrium and thus have an optimal payoff as presented in [56] are listed, but adjusted to the development of APIs in particular. The steps aim to reveal the goals, preferences and skills of participants and then decide on the social outcome. The steps are visualised in Figure 6.1 in a simplified manner.

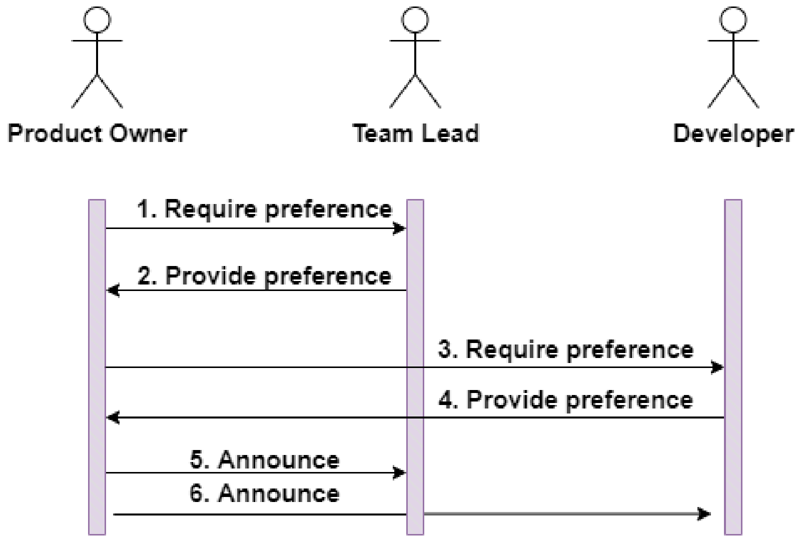

Figure 6.1.: Information exchange in Stackelberg form. The graphic is adopted from [56].

- First, team leads identify their preferences. Team leads establish a team strategy set based on possible motivators and demotivators, from both the perspective of the team leads and team members. Aligning this set with other activities of the team, the team leads prioritize the development of an API accordingly to create an incentive for the team effort.
- Team members also classify motivators and demotivators for the development of an API and inform the product owner (the one who forms the game).
- The product owner announces the given information to the teams leads and the developers sequentially.
- Based on the preferred motivators, given by leaders and followers, a decision is reached.

The motivation of development teams has been widely researched in this study, based both on literature and semi-structured interviews with professionals from different projects and different industries. These insights could be applied as a starting point as described in the design on the economic mechanism.

In section 5.4 a concept is discussed, which is based on the challenges and incentives that interview partners have shared. In chapter 7 an evaluation of the concept is discussed. Therefore, one could argue that the concept is to some extent based on this design mechanism. The actual application of the concept is the missing point from the list above and is part of future work. However, this study shows that the design mechanism presented in this chapter is applicable to studying the incentives of API development in an organisation.

7. Evaluation

In this section the quality and credibility of the results is discussed. To estimate how well the concept in section 5.4 of this study captures challenges and incentives in API development and maintenance, a follow-up round of interviews is conducted, where participants are asked to further comment on the recommendations additionally to the questions of the initial interview questions.

7.1. Evaluation Results

A total of 5 interviews included an evaluation section of the recommendations. Three of the participants are working as software architects, one is a team lead, and another is a data scientist. All participants work within the organisation, which is the main collaborator in the scope of this thesis. The recommendations, as stated in the interviews, can be found in A.2.

7.1.1. Data Sharing

When asked about the need of a platform for data sharing, participants overall support the idea and have made a few remarks. A few projects that have a similar approach and are currently ongoing in the organisation have been listed. However, a concern would be that one could lose the control over the data, in case all of it is gathered on one platform. Thus, data would be exposed at risk because of the size of the organisation. Another concern in regard to the volume of the data is that a platform would become overbuilt and a possible solution to this would be to distribute the data by domains. This idea of distribution has been supported by further participants and has been mentioned as a concern in the first interview round, making it an important point to take into consideration. A further advantage of distributing the data is the data latency.

7.1.2. Speed

Self-service for customers with little technical background is evaluated positively especially for data that is not too complex. The reason behind is APIs that provide more complex data are connected to behavior and are not easy to understand.

Self-service is a vital point when it comes to projects with external partners. It is also important that the API documentation and the API specification are on the same version.

Further remarks made for the concept of speed is that there has to be API user experience design. Customers should be able to understand if an API is relevant to them. It should also be included a way for the customer to understand the sequence in which an API works in a complex system.

7.1.3. Communication

When discussing the efficient communication between the development team and the stakeholders, the leading aspect, that interviewees have had in mind, is communication towards high quality interfaces. From their experience, informants have said that it is the case that team members should communicate frequently, but prefer not to, conditioned on their expertise. This could strongly hinder the creation of an API that is intended as a partner or as a public one. However, the idea that communication is important has been overall supported. Especially the design phase of the API development is where there should be real feedback from users and where the specification of APIs should be cleared. New requirements should also be well-communicated and since most teams nowadays work in an agile fashion, communication should be reoccurring.

Overall, the quality of the communication and the way it is organised depends on the team. To assure that products are usable, the developers that are consumers of APIs should be taken into account. A lot of user stories should be collected in the design phase for a good API to be created. In case the API is not created for an existing use case, potential clients should be contacted.

From working experience, it has been stated, that some development teams make internal reviews, pre-versions and workshops in order to ensure high quality APIs. An important factor, that has not been stated before is that when new members come to a team, the inner-organisation of the team restarts until the people within the team adjust to working together. This is not only critical, for the development of new APIs, but for their maintenance as well. An existing API of good quality could easily be made complex by implementing new requirements into existing logic that do not fit well into that logic.

7.1.4. Top Management Initiatives

The main focus of top management initiatives is creating awareness within an organisation of the key role of APIs as part of the strategic software assets of a company. While most comments on this point have been positive, an opinion has been expressed that such initiatives might not be needed, because employees are aware of APIs' advantages and their use is part of good software development practices.

On the other hand, the majority of people evaluating this idea, agree on the importance of creating awareness, especially when talking about APIs that are to be used outside of the company. Once such an API is published, the existing implementation cannot be changed.

Having APIs that are not easy to use, would make people stop using the software of that company. Thus, awareness is vital for standardizing the process and should occur at domain level if not at organisational.

Further, the awareness for API gateways has been covered.

> "API gateways are pretty good. So I think that as architects, as developers, we need to be aware of this. We need to understand about the pros and cons." [A8]

Another point that has been brought up, when commenting on top management initiatives, is raising the awareness of collaboration within the organisation.

7.1.5. Project Organisation

In general, the participants in the evaluation agree with this proposition. Thinking domain-driven is a point that one of them also further commented, namely in going from the explicit RESTful interfaces to event-driven design. It has been stressed on the fact that even experienced API developers are challenged by designing an API that uses eventing. The change in this direction is, however, necessary because event-driven API architectures have various advantages and become mandatory for specific use cases, e.g. in applying machine learning algorithms in real time, as one interviewee explained. Although ownership culture often lies within the values of big enterprises, people agree that more light should be shed into that direction.

7.1.6. Managed API Gateway

When talking about a managed API gateway, people have not only agreed that this is a meaningful recommendation, but further expressed different important aspects about them.

> "We see a lot of API orchestration, that is done via API gateways. And API gateways are also very popular now [...] as a way to realize digital platforms. Everybody is creating some or the other value-added services in the business world, in the research world. People want their services to be discoverable by others, to be used by others, to put together contact specific solutions. Because what I may create may address a very small problem. But the customer has really bigger problem. And we need a combination of such services to target that problem. These are really cases that keep coming up in the real world." [A8]

It has been further stated that API gateways may become a design pattern.

Other advantages listed are that an API gateway is a way to bring together multiple cloud platforms, that such a gateway will be necessary to ensure the security of the data. Further, open questions were also mentioned, such as how would such a solution work in a real-time system and what should the exact functionality of the gateway be.

7.1.7. Guidelines and Best Practices

The proposition of having best practices and standards to be followed within an organisation has received acceptance to some extend with some important remarks. Some say that there are good guidelines in the software community that is outside the boundaries of the organisation and in the open source community. Standardization might not be applicable due to language specifics, as well as the fact that APIs can be specific to a given use case. Others agree that this is relevant, that standardized documentation with examples is necessary as well as a lot of API governance to ensure, for example backward compatibility. Code reviews has also been mentioned as an advantageous element of API development.

Projects given as positive examples, that have been described in the interviews, all provide reviews. Thus, a extensions of this recommendation would be to conduct such reviews. They are not only needed to ensure a certain level of quality of the API and backward compatibility, but also estimate the need of the API in an already existing system, its meaningfulness.

7.1.8. Development Tools and Environment

The importance of having an environment where developers can publish quickly and securely and have tools for API development and maintenance has been differently evaluated due to the work specifics of the participants. Those, who work in research and development, do realize that there are such tools and it would be relevant to raise the awareness of standards. Others, that work with productive systems, say a lot is currently missing, such as tools for testing, for mocking, for module concept development. Developers lack knowledge how to maintain an interface even when the API is well-designed and implemented, e.g. how to go from on-premise to cloud solutions and how to extend an interface with new requirement that do not fit the existing implementation.

More experienced professionals see the need of trainings for teams, that are project specific and focus on real projects, rather than just concepts because concepts are not so well applied in practice, when learned only in theory.

7.1.9. Monetization Strategy

An important remark to the proposition of developing a monetization strategy for providing APIs to customers is that the participants in the evaluation phase have primarily experience in projects, where APIs have been used internally. Therefore, they see monetization as an important step, but think that there should be some initial awareness created around that topic and the specifics should be designed to the project accordingly.

8. Conclusion

In this chapter final remarks to the study are made. First, the results of the work presented in this thesis are summarized, followed by limitations and an outlook for future work.

8.1. Summary

A summary of each of the defined research questions, as described in chapter 1, is presented below:

RQ1: What are the challenges for providing partner APIs in internal solutions?

The research on challenges for opening internal systems via partner APIs shows that there are different obstacles at each step of the API life cycle. The strategy for providing APIs is concerning both software engineers and management roles. The developers' education should be taken into consideration and the overall organisation of teams has to be planned in a way that new tasks are properly addressed. API design holds many challenges documented in literature and discussed in the semi-structured interviews. The main concern, when designing a partner API, is balancing between the creation of a generic interface and one that is specific to a single use case. Writing documentation that is clear to follow and easily to understand is challenging for many. Other topics, such as API maintenance and monetization strategies, that come later on in the API life cycle hold further challenges. Technical obstacles, like versioning with the introduction of breaking changes as little as possible and providing the right security measurements, and organisational ones, like deciding on a monetization strategy, are faced by professionals of this study.

RQ2: What are existing incentive mechanisms motivating teams to provide partner APIs?

An important result for this research question is that there is very little academic literature about existing incentives for providing partner APIs. Therefore, other software development topics have been studied and rationales about the application of existing incentive mechanisms from them has been given. Gamification, an approach widely applied for motivating developers, is not seen as suitable for API development. Other incentives, e.g. showing projects that are successful and creating awareness, could find their application in large IT organisations in the context of API economy.

RQ3: What are incentives for providing APIs or API platforms in a large IT organization?

Surprisingly, little has been identified as a direct incentive for developing and maintaining APIs in practice. Professionals consider the technical advantages of APIs as a leading motivator as to why they incorporate them in their solutions. The granularity of APIs enables experts to reduce the complexity in existing legacy systems and maintain those easier. They see the potential to better optimize systems and to introduce new features with less effort. Interview partners have talked about the transparency as an incentive for developers. The better understanding of business processes through APIs and the direct overview of the consumption of their products are factors that contribute to the future evolution of development teams.

RQ4: What are recommendations of action for incentivizing developers and architects to provide partner APIs?

In total nine recommendations of action are proposed in this thesis. Some are providing a platform for data sharing and a managed API gateway. Essential is to help development teams by establishing guidelines and ensuring they have the needed tools and environment for API development and maintenance. In the aspect of working with partners, it is important to give consumers access to valuable assets in a self-service manner and to think about monetization strategies. Finally, at every organisational level, some management changes are advisable. Top management initiatives support an overall digitalization strategy, where APIs are in focus. Projects should be reshaped so that teams start thinking domain-driven and as an ecosystem player. Also, the communication between all people involved in the process should be planned, so that better efficiency is ensured.

8.2. Limitations and Future Work

Different limitations impact the results of the study. Many interview partners of the semi-structured interviews and the exploratory phase, conducted in this thesis, are working at the main industry partner. A few other professionals from three other companies are included. However, the limited number of participants and projects covered in the study, makes the generalization of the findings hard. The lack of literature resource that match the topic exactly has further resulted in limitations of the results.

Another limitation is the time frame for the study. In the given time, the recommendations of actions have been defined. A longer time period would have allowed for a more detailed evaluation of the concept and also for the usage of it in practice.

It is important to consider that the results of the thesis are influenced by the technical specifics of RESTful APIs as they are widely used in the studied projects. In general, many informants have experience in working with private APIs and less with partner APIs. The proposed strategies could be further refined according to the API types.

Last, many of the studied APIs are created on demand. APIs are usually coming from specific use cases and it is hard to incentivize someone to create an API without having a use case at hand. Having an existing use case directly identifies the potential customers of the interface and the foreseen resource for its creation and maintenance are easily justified.

Consecutive work of the findings of this thesis would be towards overcoming the limitations described above. The realization of the recommendations of action in an organisation will reveal their strengths and flaws and will produce base for further refinement. A practical setting will define which recommendations contribute most to the incentivization of development teams. Based on different businesses, the business value chain might vary, and the needs of partners might differ. Future work is foreseen in the adjustment of the concept to such cases and its adoption to the structures of different companies. The concept should evolve so that organisations can effectively utilize and offer services, generate new revenue streams and new business models.

A. Appendix

A.1. Questionnaire for Requirement Interviews

Introduction:

Thank you for accepting the invitation to participate in an interview to the topic "Identification and Evaluation of Incentive Mechanisms for Opening Internal Systems Via Partner APIs".

The master thesis is in cooperation with Siemens (CT, Research in Digitalization and Automation, Software and System Innovation, Architecture Definition Management/CT RDA SSI ADM-DE) and TU Munich, the chair for Software Engineering for Business Information Systems.

With this interview I aim to understand your position about API management challenges and existing and possible motivating factors when it comes to API development and maintenance. The research topic concentrates more on the social aspect of API management, in the roles of the people involved in the process rather than the technical one.

A bit more information about my research topic – APIs have become an important aspect of today's digital economy. Their ability to provide access to data, services and processes has been recognized as a valuable advantage from many organizations. My goal is to, first, understand the requirements for a possible incentivizing process and second, try to create one.

It would be interesting to know what the current situation at <name-of-company> is and to understand what are driving forces for API development and maintenance.

From this point on this session will be recorded. If you have any disagreement to this, please state it now.

General:

1. Which of the roles below is applicable to you (developer/project owner/team lead/architect)?
2. How many years have you been practicing this role?

Overview of the API-related project(s):

1. Briefly, what API related projects did you work on?
   a) Why: What was the reason to develop an API?
   b) What is the goal of the API? (Who is using it?)
   c) What is the current status of the API?
   d) What are future plans?
2. How was the API designed?
   a) Was there a defined process? Why?
   b) Did you use guidelines?
3. What challenges did you face?
   a) Why?
   b) API life-cycle? (e.g. during design, implementation, deployment, monitoring, deprecation)
4. How did you overcome those challenges?
   a) Did it work? Why?
   b) What would you do different now if you could start over?
5. Which incentive mechanisms would have helped solve the challenges?
   a) Would incentivizing help at all? Why? Why not?
   b) Which mechanisms would work?
6. What is motivating to you for API management?
   a) Why?

A.2. Questionnaire for Evaluation Interviews

Recommendations to evaluate:

1. Platform for data sharing - data description, data owner, how to access the data
2. Communication - plan efficient communication between stakeholders and developers to get feedback on time. Work together with the business/person with understanding of the domain on the design
3. Top management initiatives - create awareness within the organization
4. Project organization - organize the people in a project-based team; think domain-driven; think as a player within an ecosystem; encourage ownership culture.
5. Managed API Gateway - key point is to enable easy authentication; others are e.g. availability, security, version control.

6. Guidelines - create best practices and standards to be followed within the organization; keep re-usability in mind in the design phase
7. Development tools and environment - enable developers to publish quickly, securely and in a managed way; provide tools for API development/maintenance
8. Monetization strategy - readily designed by the business unit
9. Speed - key factor for both developers and customers (enable developers to create APIs fast and self-service for customers with little technical background)

List of Figures

List of Tables

Bibliography

[1] R. Narain, A. Merrill, and E. Lesser. *Evolution of the API economy Adopting new business models to drive future innovation IBM Institute for Business Value*. Tech. rep. 2016. URL: `https://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03759usen/gbe03759-usen-03_GBE03759USEN.pdf`.

[2] M. Careem. "Building an API Strategy Using an Enterprise API Marketplace". In: June (2017). URL: `https://wso2.com/whitepapers/building-an-api-strategy-using-an-enterprise-api-marketplace/`.

[3] *The API Mandate – Install API Thinking at your Company*. URL: `https://api-university.com/blog/the-api-mandate/),`.

[4] S. Willmott, G. Balas, and D. Weiss. "Winning in the API Economy". In: *Journal* (2013), p. 69. URL: `http://www.3scale.net`.

[5] K. Lee and N. Ha. "AI platform to accelerate API economy and ecosystem". In: *International Conference on Information Networking*. Vol. 2018-Janua. IEEE Computer Society, Apr. 2018, pp. 848–852. ISBN: 9781538622896. DOI: `10.1109/ICOIN.2018.8343242`.

[6] R. Malcolm, C. Morrison, T. Grandison, S. Thorpe, K. Christie, A. Wallace, D. Green, J. Jarrett, and A. Campbell. "Increasing the accessibility to Big Data systems via a common services API". In: *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*. Institute of Electrical and Electronics Engineers Inc., Jan. 2015, pp. 883–892. ISBN: 9781479956654. DOI: `10.1109/BigData.2014.7004319`.

[7] S. Kubler, J. Robert, A. Hefnawy, K. Främling, C. Cherifi, and A. Bouras. "Open IoT Ecosystem for Sporting Event Management". In: *IEEE Access* 5 (2017), pp. 7064–7079. ISSN: 21693536. DOI: `10.1109/ACCESS.2017.2692247`.

[8] L. Weir. *Enterprise API Management: Design and deliver valuable business APIs*. Packt Publishing Ltd, 2019. ISBN: 1787285618.

[9] M. Medjaoui, E. Wilde, R. Mitra, and M. Amundsen. *Continuous API Management: Making the right decisions in an evolving landscape*. O'Reilly Media, 2018. ISBN: 1492043524.

[10] S. Keele. "Guidelines for performing systematic literature reviews in software engineering". In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE* (2007).

[11] M. Wiesche, M. C. Jurisch, P. W. Yetton, and H. Krcmar. "Grounded Theory Methodology in IS Research". In: 41.3 (2017), pp. 685–701. URL: `http://www.misq.org`.

[12] E. Herranz, R. Colomo-Palacios, A. de Amescua Seco, and M. L. Sánchez-Gordón. “Towards a gamification framework for software process improvement initiatives: Construction and validation”. In: *Journal of Universal Computer Science* 22.12 (2016), pp. 1509–1532. ISSN: 09486968.

[13] D. Jacobson, D. Woods, and G. Brail. *APIs: A Strategy Guide*. 2012, pp. 4–46. ISBN: 9781449308926.

[14] A. Nordic. *Developing the API mindset: a guide to using private, partner, & public APIs (2015)*.

[15] I. Hammouda, E. Knauss, and L. Costantini. “Continuous API Design for Software Ecosystems”. In: *Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015*. 2015, pp. 30–33. ISBN: 9781479919345. DOI: `10.1109/RCoSE.2015.13`.

[16] *Kong: Next-Generation API platform for Microservices*. URL: `https://konghq.com/`.

[17] S. Gadge, P. Architect, V. Kotwani, and S. Engineer. “Microservice Architecture : API Gateway Considerations”. In: (2017), p. 13. URL: `https://www.globallogic.com/paper/microservice-architecture-api-gateway-considerations/`.

[18] *Full API lifecycle management: A primer - Red Hat Developer*. URL: `https://developers.redhat.com/blog/2019/02/25/full-api-lifecycle-management-a-primer/`.

[19] *The Importance of Loose Coupling in REST API Design*. URL: `https://dzone.com/articles/the-importance-of-loose-coupling-in-rest-api-desig`.

[20] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour. “A brief survey of software architecture concepts and service oriented architecture”. In: *2009 2nd IEEE International Conference on Computer Science and Information Technology*. 2009, pp. 34–38.

[21] Yale Yu, H. Silveira, and M. Sundaram. “A microservice based reference architecture model in the context of enterprise architecture”. In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. 2016, pp. 1856–1860.

[22] *Design patterns for modern web APIs*. URL: `https://blog.feathersjs.com/design-patterns-for-modern-web-apis-1f046635215`.

[23] L. Murphy, T. Alliyu, A. Macvean, M. B. Kery, and B. A. Myers. “Preliminary Analysis of REST API Style Guidelines”. In: *PLATEAU’17 Workshop on Evaluation and Usability of Programming Languages and Tools* (2017), pp. 1–9. URL: `https://codeplanet.io/principles-good-restful-api-design/%20http://www.cs.cmu.edu/~NatProg/papers/API-Usability-Styleguides-PLATEAU2017.pdf`.

[24] Zalando SE. *Zalando RESTful API and Event Scheme Guidelines*. 2017. URL: `https://opensource.zalando.com/restful-api-guidelines/%20https://zalando.github.io/restful-api-guidelines/#100`.

[25] *9 Best Practices to implement in REST API development*. URL: `https://www.merixstudio.com/blog/best-practices-rest-api-development/`.

[26] A. Mora, D. Riera, C. Gonzalez, and J. Arnedo-Moreno. "A Literature Review of Gamification Design Frameworks". In: *VS-Games 2015 - 7th International Conference on Games and Virtual Worlds for Serious Applications*. Institute of Electrical and Electronics Engineers Inc., Oct. 2015. ISBN: 9781479981021. DOI: 10.1109/VS-GAMES.2015.7295760.

[27] P. Hagglund. "Taking gamification to the next level". In: (2012), p. 37.

[28] I. Blohm and J. M. Leimeister. "Gamification: Design of IT-based enhancing services for motivational support and behavioral change". In: *Business and Information Systems Engineering* 5.4 (Aug. 2013), pp. 275–278. ISSN: 18670202. DOI: 10.1007/s12599-013-0273-5. URL: http://freshdesk.com/gamification-of-support-help-desk/..

[29] M. Tennenholtz. "Game-theoretic recommendations: Some progress in an uphill battle". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. Vol. 1. 2008, pp. 13–19. ISBN: 9781605604701. URL: www.ifaamas.org.

[30] R. B. Myerson. *Game theory*. Harvard university press, 2013. ISBN: 0674728610.

[31] M. Grechanik. "Analyzing software development as a noncooperative game". In: Institution of Engineering and Technology (IET), July 2006, pp. 29–34. DOI: 10.1049/ic:20040282.

[32] M. Yilmaz and R. V. O'Connor. "A software process engineering approach to improving software team productivity using socioeconomic mechanism design". In: *ACM SIGSOFT Software Engineering Notes* 36.5 (2011), pp. 1–5. ISSN: 0163-5948. DOI: 10.1145/2020976.2020998. URL: http://doi.acm.org/10.1145/2020976.2020998.

[33] H. v. Stackelberg. "Theory of the market economy". In: (1952).

[34] H. Jiang and J. Ruan. "The stackelberg power control game in wireless data networks". In: *Proceedings of 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, IEEE/SOLI 2008*. Vol. 1. 2008, pp. 556–558. ISBN: 9781424420131. DOI: 10.1109/SOLI.2008.4686458.

[35] J. W. Moran and B. K. Brightman. "Leading organizational change". In: *Career development international* 6.2 (2001), pp. 111–119. ISSN: 1362-0436.

[36] R. T. By. "Organisational change management: A critical review". In: *Journal of Change Management* 5.4 (Dec. 2005), pp. 369–380. ISSN: 1469-7017. DOI: 10.1080/14697010500359250. URL: https://www.tandfonline.com/doi/full/10.1080/14697010500359250.

[37] R. Gill. "Change management–or change leadership?" In: *Journal of Change Management* 3.4 (Dec. 2002), pp. 307–318. ISSN: 1469-7017. DOI: 10.1080/714023845. URL: https://www-tandfonline-com.eaccess.ub.tum.de/doi/abs/10.1080/714023845.

[38] L. Murphy, M. B. Kery, O. Alliyu, A. Macvean, and B. A. Myers. "API designers in the field: Design practices and challenges for creating usable APIs". In: *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*. Vol. 2018-Octob. 2018, pp. 249–258. ISBN: 9781538642351. DOI: 10.1109/VLHCC.2018.8506523.

[39] S. Andreo and J. Bosch. "API Management Challenges in Ecosystems". In: *International Conference on Software Business*. Springer, 2019, pp. 86–93.

[40] N. Baddoo and T. Hall. "Motivators of Software Process Improvement: An analysis of practitioners' views". In: *Journal of Systems and Software* 62.2 (2002), pp. 85–96. ISSN: 01641212. DOI: 10.1016/S0164-1212(01)00125-X. URL: www.elsevier.com/locate/jss.

[41] E. Haruvy, F. Wu, and S. Chakravarty. *Incentives for Developer's Contributions and Product Performance Metrics in Open Source Development: an Empirical Exploration*. April. 2005. ISBN: 1075-2730. DOI: 10.1176/appi.ps.57.7.1035. URL: http://115.111.81.83:8080/xmlui/handle/123456789/6277.

[42] P. Fremantle, J. Kopecký, and B. Aziz. "Web API management meets the internet of things". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9341. Springer Verlag, 2015, pp. 367–375. ISBN: 9783319256382. DOI: 10.1007/978-3-319-25639-9{_}49.

[43] M. P. Robillard and R. Deline. "A field study of API learning obstacles". In: *Empirical Software Engineering* 16.6 (2011), pp. 703–732. ISSN: 1382-3256.

[44] H. G. Afridi. "Empirical investigation of correlation between rewards and crowdsource-based software developers". In: *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*. 2017, pp. 80–81. ISBN: 9781538615898. DOI: 10.1109/ICSE-C.2017.149.

[45] E. Schindler. *Building a developer ecosystem: What vendors do to attract you to their platforms*. 2013. URL: https://www.itworld.com/article/2704738/building-a-developer-ecosystem--what-vendors-do-to-attract-you-to-their-platforms.html.

[46] D. R. Agrawal. "If Open Source Code Is a Public Good, Why Does Private Provision Work (Or Does It)". In: *LBJ J. Pub. Aff.* 18 (2005), p. 55.

[47] "The future of enterprise applications". In: AMR Research, pp. 1–1. ISBN: 9780978592813. DOI: 10.1109/icis.2013.6607805. URL: https://books.google.de/books?id=3O1Z7boX8ZEC.

[48] J. A. Halderman. "To strengthen security, change developers' incentives". In: *IEEE Security and Privacy* 8.2 (Mar. 2010), pp. 79–82. ISSN: 15407993. DOI: 10.1109/MSP.2010.85.

[49] *Evans Data Corporation | Five PitFalls to Avoid in a Developer Program*. URL: https://evansdata.com/reports/viewRelease.php?reportID=36.

[50] S. Liu. "Research on Token Incentive Mechanism of Open Source Project - Take Block chain Project as an Example". In: *IOP Conference Series: Earth and Environmental Science*. Vol. 252. 2. IOP Publishing, 2019, p. 022029. DOI: 10.1088/1755-1315/252/2/022029. URL: http://dx.doi.org/10.1088/1755-1315/252/2/022029.

[51] C. R. Ramesh, K. N. Prasad, H. H. P. K. Bhuravarjula, and V. V. D. N. Krishna. "Customer Relationship Management System with {USCM-AKD} Approach of {D3M}". In: *International Journal on Computer Science & Engineering* 1.6 (2010), pp. 2036–2040. ISSN: 09753397. URL: https://www.researchgate.net/publication/49620491_Customer_Relationship_Management_System_with_USCM-AKD_Approach_of_D3M.

[52] R. Meier. "Business Model for a Developer Eco-System". PhD thesis. Swiss Research Institute of Small Business and Entrepreneurship, 2017, p. 58.

[53] R. D. Morey, C. D. Chambers, P. J. Etchells, C. R. Harris, R. Hoekstra, D. Lakens, S. Lewandowsky, C. C. Morey, D. P. Newman, F. D. Schönbrodt, W. Vanpaemel, E.-J. Wagenmakers, and R. A. Zwaan. "The Peer Reviewers' Openness Initiative: incentivizing open research practices through peer review". In: *Royal Society Open Science* 3.1 (Jan. 2016), p. 150547. ISSN: 2054-5703. DOI: `10.1098/rsos.150547`. URL: `https://royalsocietypublishing.org/doi/10.1098/rsos.150547`.

[54] *Enterprise*. URL: `https://rapidapi.com/enterprise`.

[55] *AWS Data Exchange - Access Third-Party Data In The Cloud - Amazon Web Services*. URL: `https://aws.amazon.com/data-exchange/`.

[56] M. Yilmaz, R. V. O'Connor, and J. Collins. "Improving software development process through economic mechanism design". In: *Communications in Computer and Information Science*. Vol. 99 CCIS. 2010, pp. 177–188. ISBN: 3642156657. DOI: `10.1007/978-3-642-15666-3{\_}16`.

[57] Y. Dittrich, C. Floyd, and R. Klischewski. *Social Thinking–Software Practice*. Mit Press, 2002. ISBN: 0262042045.

[58] E. S. Yu. "Social Modeling and i". In: *Conceptual modeling: Foundations and applications: Essays in honor of John Mylopoulos*. 2009, pp. 99–121.