Computer Science • 25(4) 2024 https://doi.org/10.7494/csci.2024.25.4.5423

Marcus Hilbrich
Ninon De Mecquenem

MICROSERVICES, A DEFINITION ANALYZED BY SSMACH

Abstract *Managing software artifacts is one of the most essential aspects of computer science. It enables to develop, operate, and maintain software in an engineer-like manner. Therefore, numerous concrete strategies, methods, best practices, and concepts are available. A combination of such methods must be adequate, efficient, applicable, and effective for a concrete project. Eelsewise, the developers, managers, and testers should understand it to avoid chaos. Therefore, we exemplify the ßMACH method that provides software guidance. The method can point out missing management aspects (e.g., the V-model is not usable for software operation), identify problems of knowledge transfer (e.g., how is responsible for requirements), provide an understandable management description (e.g., the developers describe what they do), and some more. The method provides a unified, knowledge-based description strategy applicable to all software management strategies. It provides a method to create a minimal but complete description. In this paper, we apply ßMACH to the microservice concept to explain both and to test the applicability and the advantages of ßMACH.*

Keywords ßMACH, microservices, software artefact management, process model, architectural style

Citation Computer Science 25(4) 2024: 1–32

1. Introduction

Managing software artifacts is one of the most essential aspects of computer science. The question is how to develop, operate, and maintain software [16, 66, 75, 76, 80, 81]. Scientists and industry give different (partial) answers to this question: software process models [1, 26, 47, 69, 70, 73, 74], programming paradigms [4, 8], change management [3, 9, 11, 14, 24, 32, 35, 41, 53, 86], best practices [10, 19, 42–44, 54, 85], and many more [7, 15, 18, 25, 28, 48, 79, 84]. So, to tackle one challenge, we have various solution strategies, methods, description languages, and suggestions present. We neither have a uniform solution strategy nor a description language. For managing concrete software, e.g., in a project [67, p. 1] [70, p. 3,5,14], software developers, managers, testers, and reviewers are deciding and understanding/learning a management strategy. Thus, the open question in concrete is:

How to review if a software artifact management strategy is suitable ahead of the beginning of the management?

In principle, a management strategy has to cover at least all relevant aspects of software management. So, the V-Model is insufficient if a project requires maintaining and operating software. A software management strategy should not cover additionals, to be minimal. So, a change management process is insufficient if the decision to create software is already done. Knowledge transfer is an additional challenge. If a Kanban-based project decides on an explicit requirement engineering, the results of the requirement engineering are included in the development process. Therefore, the requirement engineer can deliver a programmer understandable description or participate in the development process. Requirements are not present in a Scrum process. Such a process requires user stories. They also describe the same functional properties of software artifacts.

The examples describe the overall challenges. The different management strategies cover varied aspects, but a project requires managing exactly the project's aspects. All involved persons have to understand their role in the overall process. Creating and transferring (by persons, documents, and software artifacts) has to be explicit to enable management. The different software management strategies use different terms and languages, which makes them hard to understand and combine.

Our solution is a uniform, complete, minimal, and easy-to-apply software guidance. It applies to all management strategies and their combinations. ßMACH handled the challenge that an a priori a unification of the languages used in computer science is not possible [46, 91] [72, p. 319]. Nevertheless, it is possible to describe the knowledge. Examples are the description of ontology's of engineering [75, 76], the work by Popper [63, 64], or other general categorizations [76].

The ßMACH method [36, 40] provides a unification of the aspects based on a unification of management strategies, an extraction of relevant management aspects, and a systematization of these aspects. As a result, the ßMACH method enables a software developer to check and specify a concrete management strategy with minimal effort. The management strategy becomes easily understandable to programmers, project managers, engineers, testers, etc. The ßMACH method enforces a description of all relevant aspects of software management and identifies missing ones enforce completeness. It enables review strategies, fosters improvements, is a starting point for academic discussions, and is the basis for systematic comparisons of management strategies. Therefore, it can describe all kinds of management strategies in a uniform knowledge-based language and avoids chaotic processes.

In this paper, we exemplify the ßMACH method to explain it, provide its advantages, and test its applicability. Therefore, we have to decide on a management strategy. Scrum, V-model, or Kanban are describing such management strategies. They are easily describable by the ßMACH method, so we decide on a more challenging task.

We decided on microservices. It is a concept, not a management strategy. Microservices are an answer to various scalability challenges. They enable large and complex systems by scaling the number of services and development teams. Microservices allow self-management and agile processes such as Scrum. Therefore, the developers apply the microservices concept to all parts of the overall system. It requires all developers, managers, software engineers, etc., to understand the microservice concept and to follow it, so the ßMACH-based description in this paper is helpful.

This paper focuses on using the ßMACH method and not creating the ßMACH concept. Therefore, we explain the usage of ßMACH on a toy example and analyze microservices. We decided on microservices because they are well-established and widely used in large-scale industrial applications [56,57,78]. Such systems have proven to be scalable to support several million users. Academics described them at various conferences [17, 30, 31, 34, 52, 55, 77] and discussed them heavily. Thus, microservices are an answer to actual scalability challenges and an object of academic research.

The scalability of microservices is not limited to the user load. Also, the development is scalable. Microservice systems consist of individual microservices [12, 21, 59, 90, 93]. Every service is developed and managed by one team (but a team can have multiple services). The idea of microservices is to keep a team small, as described by Levis and Fowler [50]. Most importantly, the teams and the microservices stay small, even if the overall system can scale. It scales by adding additional microservices and teams.

The small teams provide a set of advantages like less management overhead. The team members are more productive in a flat hierarchy, and agile software management processes are easy to learn. The microservices and teams are independent of each other. Thus, the overall management overhead is reduced. Nevertheless, too many requests based on service communication or other aspects can hinder the productivity of a small team [54]. It is the reason for demanding service and team isolation. It is an essential factor of microservices. Based on the isolation, we call our microservice definition a strong one, as opposed to code size, the number of team members, or the used technology as the basis [2, 5, 58, 82, 89].

In a microservice system, even if the teams are small and self-managed [23, 45, 70], a minimum set of rules has to be set. The teams should not break the microservice system, e.g., by building interfaces to other microservices. In addition, all management aspects need a description. Software management includes creating, improving, deploying, and operating software artifacts [74]. Also essential are the documentation and communication of the teams. All members need to understand and agree to the process to avoid conflicts.

ßMACH [36, 40] documents and defines a software management process. It aims to check the management strategy to support all relevant management aspects and avoid unnecessary management. ßMACH is minimalistic, based on scientific groundwork and an ontology of key management aspects. It provides engineer-like systematics. In this paper, we give a ßMACH protocol to accomplish all significant aspects of software management in a microservice team. It describes how to create such a protocol. So you can adapt it to your concrete process. As a result, based on the ßMACH protocol, we can demonstrate how the independence of microservices is a solution to many management aspects.

We organize the paper as follows: An introduction to ßMACH in Sec. 2, to give the fundamentals of the scientific method. In Sec. 3, the microservice system is defined and explained. Sec. 4 describes a use case. Then (Sec. 5), we explain how to fill the ßMACH protocol (perform the ßMACH method). With the ßMACH protocol, we can provide observations on the microservice-based process to manage, and we will analyze microservices (Sec. 6). We close with a conclusion (Sec. 7).

2. A short explanation of ßMACH

The ßMACH[1] method is an approach to define and plan a software-management process, e.g., a software development project. Therefore, ßMACH defines the management process, gives additional context (meta-information), and describes how to cover essential aspects of software management (As groundwork, see, e.g., [4, 6, 9, 13, 20, 22, 27, 29, 48, 51, 60, 62, 73, 74, 87]). The key aspects are based on an ontology of software engineering and software management strategies and are described based on the vocabulary of knowledge management.

The ßMACH protocol consists of three parts: the definition of management processes, meta-information, and descriptions of the key aspects. A team should fill in a protocol for each separate management process. The meta-information defines the team, the filler of the protocol, and additional parts. The definition of *Our Team* and *Cooperating Teams* are essential for understanding the protocol. Our team is the group of persons who directly manage the software. In our example, the team manages one microservice of the overall microservice system. Cooperating teams are other teams that are intensively involved. An example would be a dedicated testing team. In our case study, there is no such team.

[1] Systematic Software Management Approaches Characterization Helper; ß is the German Eszett. You can read and pronounce it as “ss”.

By defining the management process (Fig. 4 and 1), we provide the guidelines to plan the management. It can be short and link to additional documents. For instance, we can reference the Scrum Guide [70] in the case of Scrum-like management. The definition should be easy to understand by the target audience. Usually, this audience is the team. In this case, the readers of the paper. The definition should be in numbered bullet points. So it can efficiently describe the key aspects.

ßMACH defines three groups: *our team*, *cooperating teams*, and *externals*. *Our team* is the group that manages the software, so they have to deliver and operate a microservice (of the microservice system). Also, *our team* defines and learns the management process in the ßMACH protocol. *Cooperating teams* are other teams that *our team* can or must cooperate with. *Our team* can not define how *cooperating teams* work (they do their own management). *Our team* can change the agreements with them during the management process. Such teams are, e.g., teams in the same organization. Because *our team* is working with such teams, ßMACH calls this *Internal*. The last group is called *External*. *Our team* can not directly influence such parties, e.g., contract partners. *Our team* has a defined contract and has to follow it. Another example is a provider of a library or end-users who use the microservice.

The ßMACH protocol organizes the description of the key aspects of software management in a table. Fig. 3 provides an example. The columns define the different aspects of knowledge and information management. It includes who is doing (column *Roles*), what needs to be known to perform the process (column *Process Knowledge*), and how is the product or aim of the process (column *Product Knowledge*). This part of ßMACH follows the idea that a product or artifact is created and managed by actors/persons/roles in a process. [74]. In addition, ßMACH points out if a piece of knowledge is not present at the beginning (column *Demanded Knowledge*). Clarifying which knowledge is required is essential as the process needs to find a solution to acquire it during the management. The last column is called *Process Information*, which defines which information has to be provided by the management process, e.g., working hours for billing, the results of meetings, and delivery protocols.

The rows define the product aspects and the party that influences the aspects. Previously, we gave the type of parties. Our team has external parties (marked by the term *Outside*), and our team has cooperating teams (marked by the term *Inside*). Our team is present by the table and needs to conduct the management process based on the key aspects, given in the table.

The rows in the table represent the *Product Properties*. The artifacts our team has to develop/manage. *Interfaces* are the definition of (technical) interfaces of our artifacts to communicate with other systems. *Dependencies* describe everything our team demands to get from others. *Responsibilities* give what our team needs to provide to others (e.g., based on contracts or regulations). Each of the four aspects is present as internal and external. So, ßMACH defines interfaces to cooperating teams that can be discussed and adapted based on the project needs and fixed interfaces to external parties. These aspects cannot be influenced directly by our team.

The last row is *External Artifacts*. It describes that an artifact is taken from another party and included in our project. It is copied (e.g., use an open-source library). As a result of copying, it is irrelevant whether it is from a cooperating team or external. Nevertheless, an external artifact needs management. Our team needs to know how the artifact works, how we will use it, and other consequences (e.g., based on licensees) the team has to consider.

In addition, ßMACH defines relations of different aspects, abstracted as cells in the table. One aspect can require another, so knowledge transfer or transformation is required. A provides relation expresses that an aspect does not need active management. The aspect is handled/provided as a consequence of managing another one. For example, forbidding the usage of external artifacts provides a solution to all related management aspects by avoiding any need for management.

To exemplify the ßMACH method we provide a toy example in the appendix. The toy example provides additional explanations for all its parts. Thereby, it is possible to look up what to fill in the protocol and have a very simple example of a filled protocol. For this paper, we split the protocol into parts to support printing. The original protocol consists of an A4 page for the definition of the management plus meta-information and A3 pages for the description of the key aspects. We give the management definition in Fig 4, and the meta-information in Fig 5. The table with the key aspects has an initial explanation. We present it in Fig. 6. The original protocol provides one table with all key aspects. In this paper, we split it into three parts, presented in Fig. 7, 8, and 9. For the toy example, we use blue text to visualize everything we (or the team) filled in. The hints are presented in gray text. You can download the ßMACH protocol at https://doi.org/10.5281/zenodo.10992007 to print it in large or zoom in to read all the details.

The toy example is the following. A company has a magical box to create software. So, your challenge is to find out how to get the box. One team of the company is our team. This team wants to use the software itself. Thus, communication, dependencies, management goals, etc., are extremely reduced compared to a realistic project. The magical box can be interpreted as a simplification of outsourcing. So, payment circumstances, problems with the outsourcing partner, etc., are removed from the example.

The toy examples use the different relations of key aspects. The fields in the column process information (Fig. 8 and 9) are all very similar. Based on the management definition, no information is recorded. In each case, the part 6 of the definition is referenced. Therefore, the similarity is visualized by the same background color.

The toy example provides different examples of require relations. One example is from the field inside dependencies / product knowledge to inside product properties / roles. The dependency describes the need to get the magical box. This box provides the product properties, and someone (a role in the team) needs to get the box from another team of the company. As a result, the dependency requires a role to support resolving the dependency.

An example of provide relations is, e.g., present in the row inside dependencies. The inside dependency is defined by getting the magical box. That is what you need to know to handle the dependency. So, it is in the column product knowledge. This field is resolved by a demand relation described above. In addition, the magical box does not need extended management. Because it is so easy, it does not need an additional/extra role to manage it or a process. Thus, product knowledge provides a solution for field roles and field process knowledge. In this case, it provides a solution because it can be denied to have a role or a process to manage the inside dependencies.

Later on, we describe the filling in of an ßMACH protocol in detail. So, we stay with the toy example as a self-explaining, very simplified example.

3. Strict definition of microservices

The paper aims to analyze microservices with the ßMACH method. Therefore, we defined and described microservices in general, and based on the ßMACH method. We are not describing a concrete microservice project, and we do not deny that those real-world systems need to find compromises between the strict isolation our definition demands and practical circumstances. Thus, we do not describe a concrete, but rather a preparation of a management process. It checks if the microservices concept describes all management aspects defined by ßMACH. For a real-world management process, we would need a more concrete context and an adaption to give missing descriptions in the microservice concept.

Before using ßMACH, we start with the general description: The term microservice is not well-defined: The term is used for SOAs [93] build of small services [61,83], for a realization of an organizational structure [50], as a DevOps concept [50,90], or as architectural style [12,21,50,59,93]. Our definition focuses on the strict isolation of individual services because isolation can be helpful for management processes [13,71].

In the following, we provide a clear definition of microservices. We used definitions stated before (see also [37–39]), a combination of common definitions and strategies, e.g., [12, 21, 50, 59, 93]. We present our definition as a pattern:

Name Microservices (also called Slice Service Style)
Problems to solve Solves the need for scalability concerning the system load and the number of persons/teams developing the system.
Definition The slice service style is an architectural style where the essential aspects of the system are encapsulated in services (slices, microservices, or vertical services). These services deliver functionality to end-users and have no (or minimal) dependencies on other slices of the system. It includes code-sharing, usage of interfaces, sharing of manpower, and management of, e.g., creation, deployment, and operation.
Consequences Because of the separation of slices to allow scalability, the software process model needs to be adapted or tailored. The definition of slices

influences the overall system and has to be done globally (e.g., up to the design phase of the waterfall model), while the creation and operation of the slices are independent. Thus, the (global) software process model has to support independent software development (e.g., by realizing each slice as a DevOps project) and a design or architectural phase at the beginning.

Drawbacks Because the independence of slices includes teams and persons, the structure of the organization developing the system needs to be aligned. In addition, independence reduces the knowledge transfer of the persons of different slices and hinders common reuse techniques. Especially cross-cutting concerns cannot be managed.

4. Use case

In the following, we present concrete use cases from the development team's perspective. We give examples of how the ßMACH protocol can be helpful in concrete and how it is used by the team. Therefore, we use the microservice example as a basis, but we will also point out differences to a concrete ßMACH protocol.

4.1. The external artifact question

When our development fills in the ßMACH protocol, they get to the row about external artifacts. Microservices, as a concept, do not provide a clear and commonly accepted solution strategy. As a result, our team is pointed to this challenge and needs to make a clear and informed decision. Typical answers are the following:

- To reduce the dependencies on external code, we forbid the usage of external liberties. In the ßMACH protocol, we fill in that no knowledge regarding external artifacts exists and no management process is required. We make it clear to the team members by adding "It is forbidden to use external artifacts." as an item to the management definition.
- To forbid external libraries creates new challenges. Encryption, single sign-on, and logging are forced to be re-implemented. This is a high, additional effort and very error-prone. Using established, well-tested, and continually supported libraries is a solution strategy. In such cases, it demands to know the libraries and check that the licenses are exportable. Integrating the liberties requires checking for security issues and update own microservice on demand. Therefore, it requires adding a process and role to the ßMACH protocol.

The ßMACH concept forces the team to decide how to handle external artifacts. The team decides on a strategy and avoids unwanted problems like unmanaged security issues based on outdated libraries.

4.2. Why not use another microservice?

Let us assume we have a running microservice, and our team operates and maintains it. In this situation, our team gets a new member who proposes to use the other microservices to reduce the code base and increase the functionality.

The team can refer to part 1 of the definition (Fig. 1). Thus, the new member can understand the current situation.

The ßMACH is not written in stone. If the situation changes, the protocol can be updated. In this example, it is discussed to remove part 1 of the definition. As a result, all key aspects referring to this part (Fig. 3) are part of the discussion. In concrete, cooperation with other teams has to be established and managed. It is, e.g., needed to have a plan if another service changes the interface or is temporarily unavailable.

Whether it is more effort or risk to manage the relation to other teams/microservices or to not use their services is a decision of our team. ßMACH demands to describe the plan to foster an informed decision.

4.3. The functionality of the microservice

One of the open questions in our example is the functionality of a singular microservice. In ßMACH, this is mainly a question of product knowledge. To program and maintain the microservice, our team needs the related product properties (see Fig 3). In short, the interfaces the microservice provides to the end-user define the product knowledge (the code base of the microservice). The code needs to implement the realization of the interfaces. The responsibilities (the definition of what our microservice has to provide to the end-user) define the interfaces. A chain of demands relations in the ßMACH protocol (see Fig 3) represents the knowledge transfer. The open question in the protocol is who (which role) provides the responsibilities of our service as a system's concern of the overall microservice system.

In our example, we can not answer the system's concerns at all. The concerns require a concrete system and project. Without knowing the aims, purpose, or business model of the microservice system, we can not answer. A real microservice system example has such information available, at least for the overall microservice system.

The ßMACH protocol we provide in this paper is for the development team of one microservice. Thus, from the viewpoint of this team and the corresponding ßMACH, the knowledge of the partial system's concerns (outside responsibilities) needs to be provided somehow. If we create a ßMACH protocol for another microservice and another team, we encounter the same problem. As a result, we demand additional teams that define the business capabilities of the overall microservice system. In addition, such teams separate the overall system's concerns into individual microservices [33]. To describe such a team in ßMACH is another story. It requires having an overall business strategy [13] and dividing [92] the overall business concerns into individual services.

5. Filling of the ßMACH protocol

We use the microservice definition to describe the filling of the ßMACH protocol. The first step is the discussion of the context information. Then, the definition and the aspects of management are discussed in parallel. The results are Fig. 1, 2, and 3.

5.1. ßMACH context

To fill the ßMACH protocol starts with writing down the context information. This part of the ßMACH protocol defines other parts. So, it is a good starting point. Mostly, the context is very clear and easy to fill. We know the name and the date a priori. It is the first version, so we label it as 1.0.

The ßMACH protocol is filled for a team that manages a microservice, not for the organization that manages the overall microservice system. We call the team Microservice Team A, A to indicate that other teams of this kind exist. The team needs more details than a name, so we added an explanation in Fig. 2. This also describes the artifacts. In the pattern definition (Sec. 3), the part "This includes code-sharing, usage of interfaces, sharing of manpower, and management of, e.g., creation, deployment, and operation." describes the separation and the artifacts. The part "by realizing each slice as a DevOps project" describes the different teams.

The cooperating teams are mostly independent. Thus, we could define them as external. Also, the teams belong to the same organization. The organization manages the overall microservice project. It argues against a complete disjoin. We use the system border of the microservice system as the external border. The mapping of individual microservices to teams is sufficient to describe it: all teams work on the same microservice system as cooperating teams, even if they are independent. As a result, the context information of the ßMACH protocol is present in the ßMACH protocol in Fig. 2.

5.2. ßMACH Definition and software management aspects

ßMACH is a method to define and discuss a software management process. The ßMACH protocol defines the process required by a software development team. The team is responsible for a microservice. We start with the work packages. Afterward, we describe the management process of our team.

5.2.1. Work package responsibilities

Work packages of ßMACH describe if *our team* is responsible for the development, maintenance, and improvement. The pattern-like definition of microservices (Sec. 3) mentions development, maintenance, and improvement. Development is called "creation". The "deployment" is a part of the development and/or maintenance (depending on static deployment or the system changes its deployment). The "operation" is at least part of maintenance and can include improvement. The mention of DevOps confirms that all work packages are included in the management process. Thus, the team is responsible for all work packages, and we check them in the ßMACH protocol (Fig 1).

5.2.2. Definition of the management process

We have a microservice definition (Sec. 3), but it is not a ßMACH protocol. We need a description where different parts of the definition are easy to reference. Also, each part should describe one aspect and no mixtures.

To get the definition for ßMACH, the definition from Sec. 3 is decomposed and recomposed. We can split the first sentence of the pattern-like description into parts that are candidates for the ßMACH definition:

- The naming microservice and the classification as architectural style.
- The representation of system concerns as encapsulated services.
- The services deliver functionality to the end-users.
- Services have no (or minimal) dependencies on each other.

The second sentence describes what is included in the services and is independent of other services:

- Services have a code base.
- Services have interfaces.
- Services have a team ("manpower").
- Services persist over development, maintenance, and improvement ("creation, deployment, and operation").

The naming and classification as architectural style do not give the descriptions as needed by the ßMACH protocol. In addition, we can reorder the items in the description of the system and the microservices:

I The microservice system consists of microservices.
II Microservices have no (or minimal) dependencies on each other.
III Microservices represent encapsulated system concerns.
IV Microservices are persisting over development, maintenance, and improvement.
V Microservices deliver functionality to the end-users.
VI Microservices have interfaces.
VII Microservices have a code base.
VIII Microservices have a team.

For the ßMACH protocol, it is only allowed to add needed parts to the management definitions. Thus, it is a reasonable strategy to develop the definition of the management process by answering the questions about the management aspects in the table of the ßMACH protocol. This table provides two separate parts. They are the work the team is not responsible for and the part the team needs to manage directly. In the following, we give both parts.

5.2.3. Not responsible for

The team is responsible for product development, maintenance, and improvement. Thus, we have to cross the fields in the table. So, we finished the rows of product development, product maintenance, and product improvement. We do not have to prepare for other teams to overtake the work. It is typical for DevOps-like strategies.

5.2.4. Responsible for

In the following, we have to provide the descriptions of the software management aspects and fill the table. By filling the table, we have to refer to the parts of the software management concept. As the current starting point, this part of the ßMACH protocol is empty.

We start with interfaces. (There are two rows for interfaces in the table.) We already mentioned interfaces in item VI. The interfaces are inside interfaces in case cooperating teams use the interfaces. It would be a kind of dependence that II mostly denies. Thus, the interfaces are mostly used by externals. Externals are the end-users according to V. Item III describes the purpose of the microservice. Because the pattern-like definition does not mention other communication, it needs to be offered by the interfaces. So, inside interfaces can be mostly denied. The product properties of the outside interfaces are a subset of the system concerns. For the ßMACH definition, we combined items I and II from above to part 1 of the ßMACH definition. In addition, we combine items III, V, and VI from above as part 2 of the ßMACH definition (Fig 1).

We state that internal interfaces are not present. In short, we deny them. To deny internal interfaces means we can deny the need for product knowledge. The team does not need to know anything about nonexisting interfaces. We denied the other cells in the row, too. There is no need to explore additional knowledge (demanded knowledge), no management process is needed (process knowledge), no one needs to do something (roles), and the team cannot record information about the nonexisting process (process information). In other words, based on the fact that no interfaces exist (product knowledge), no role is needed. In ßMACH, this is a provided relation. Based on the cell product knowledge, other cells in the row are filled/inferred. Provide relations are marked by an arrow and according to the coloring of the right side of the cell, as provided by the filled ßMACH protocol in Fig. 3.

To deny product knowledge based on the independence of services can be directly transferred to the rows inside product properties, inside dependencies, and inside responsibilities. The arrow for the provided relations and the coloring of the right part of the cells are the same. We applied the same argumentation to the cell product knowledge in different rows. ßMACH demands to use the same color, in this case (it is not a relation, so the left part of the cell is colored). Based on the same argumentation, we used the same color (Fig. 3). So, we finished the rows inside interfaces, inside product properties, inside dependencies, and inside responsibilities.

Part 2 (Fig. 1) of the ßMACH definition does not only define the product knowledge of outside interfaces. The interface and the concerns define the outside product properties. Part 2 defines the product knowledge of the outside responsibilities by a subset of the system concerns, too. Thus, all three fields get the same color in the ßMACH protocol (left side of the cell). In addition, we visualize that the three fields depend on each other. The concern of the system presented by the responsibility is best. The interface is just the technical and organizational presentation of the responsibility. So, it is dependent on fulfilling the responsibility. The product property is the realization of the interface. So, the fields of product knowledge in the rows outside product properties, outside interfaces, and outside responsibilities are defined.

The definition of an architectural style (Sec. 3) does not describe how and by whom the artifacts should be managed. The consequences part of the pattern-like definition is helpful. It gives the tailoring of the software process model. We start with the part that describes that DevOps projects are present for each microservice. It helps to describe additional fields. The DevOps team has to provide all needed roles, and it is small enough to manage itself. It gives the roles and the process knowledge for the rows outside product properties and outside interfaces. Because we gave all descriptions based on the DevOps team, we use the same color for all fields (left part of the cells). We present the result in Fig. 3. Now, we add the DevOps team to the ßMACH description as part 3.

The DevOps team knows how to develop and maintain the product. Thus, the DevOps team members have product knowledge (row outside product properties). In other words, the DevOps team builds and maintains the software based on their knowledge/experience (and based on the definition of the interfaces). Microservices do not give an additional knowledge object. Therefore, we extended the product knowledge cell in this row and the demand relation (Fig. 3).

Demanded knowledge (row outside product properties) does not exist. The DevOps team manages the artifacts. The outside interfaces present a definition of the product. It adds two provides relations, so everything is present.

The "consequence" section of the pattern-like definition (Sec. 3) gives the root of the concerns managed by the team. The product knowledge in the rows outside responsibilities demands it. The separation of system concerns is not described (probably given by another team). Thus, it is demanded knowledge for our team. How to obtain this knowledge is unclear. We cannot name the needed process, roles, and process information. We use question marks and red coloring. Also, the ßMACH definition is extended by part 4. The outside interfaces are (mainly) defined by the outside responsibilities. In the case of a concrete end-user, we would need additional aspects, concretization, and adaptions. The DevOps team handles these interfaces. These are two provides relations.

The column process information is not directly covered by the pattern-like definition (Sec. 3), but the concerns of the system can demand such information (e.g., accounting of used resources to benchmark efficiency). Thus, our team transfers the (description of) product knowledge to process information for the rows of outside responsibilities, outside interfaces, and outside process properties.

The pattern-like definition does not give or define the row outside dependencies (e.g., to use an external service or external artifacts). Both can be demanded or forbidden by the system's concerns. So, a demand relation exists. Otherwise, we can expect that the DevOps team manages artifacts and dependencies (similar to outside product properties and outside interfaces we give them in the already defined color). The product information depends on the system concerns (e.g., for the outside interfaces, the system concerns define also the process information directly, but we do not describe it this way). Our team does not demand additional knowledge. We expect the DevOps team to have the needed skills and knowledge. So, we finished the table of ßMACH descriptions (Fig. 3).

5.3. Filled ßMACH protocol

We separated the parts of the ßMACH protocol. Fig. 1 presents the definition, Fig. 2 the meta-data, and Fig. 3 explains the management aspects.

The ßMACH protocol has only two open, not complete answered cells (Fig. 3). The cells describe the separation of concerns of the overall system to isolated microservices. It is a challenge of microservices [49, 65, 68, 88].

The microservice definition describes many cells, especially product knowledge. The system concerns are a basis, with many relations in the ßMACH protocol. The independence of services enables answering inside related rows.

The architectural style does not fully describe roles and processes. The usage of DevOps answers such questions.

Based on the strict description of microservices (the pattern-like definition, Sec. 3), we can fill a ßMACH protocol (Sec. 5.3). Thus, the given definition covers nearly all relevant aspects of software management. The red-colored cells in the ßMACH protocol point out the open challenge of microservices to define independent concerns of the overall system.

The microservice definition (Sec. 3) holds information not present in the ßMACH protocol. Thus, parts of the definition do not describe software management aspects.

The name of the pattern-like definition is (somehow) represented in the context part of the ßMACH protocol (Fig. 2). The "problem to solve" part is not needed to fill the ßMACH protocol. This part can help to decide whether to use microservices or not. It is not in the scope of the ßMACH method. ßMACH helps to understand if all aspects of managing software artifacts are covered. It is no direct helper to decide to use a specific management method, but it can check different strategies. So, the aims of the pattern-like definition and ßMACH are different.

The "definition" sections of the pattern start with the description in an architectural style. The ßMACH protocol does not cover it. So, it describes a pattern property, like the description itself. The rest of the definition sections cover parts 1 and 2 of the ßMACH definition (Fig. 1).

The "consequences" part of the pattern-like definition gives two pieces of information. First, detail of the service separation (part 1 of the ßMACH definition). Second, the separation of concerns and services has to be realized somehow (part 4 in ßMACH). Third, individual services are realized by DevOps (part 3 in ßMACH).

The drawback section of the pattern-like definition is not represented by ßMACH. It describes problems outside our team, and ßMACH does not represent them. It is an additional different aim of the pattern-like definition and ßMACH.

Work Package Responsibilities:

- ☑ Finelizing Product Development
- ☑ Finelizing Product Maintains
- ☑ Finelizing Product Improvement

Definition of the Management Concept:

1. The microservice system consists of microservices, microservices have no (or minimal) dependences to each other.
2. Microservices represent encapsulated system concerns that are delivered via interfaces to endusers.
3. A microservice is managed by a DevOps team that provides all needed knowledge and manages itself.
4. The separation of system concerns to microservices has to be realized, how to do so is not covered by the microservice concept.

Figure 1. Definition of microservices in the ßMACH protocol. The definition gives the numbers of the bullet points/parts. (See Fig. 3 and https://doi.org/10.5281/zenodo.10992169 for more details.)

Context (User/Team/Context Information):

Name of the Filler:	Marcus Hilbrich
Represented Team (Internal Border):	Microservice Team A: one of the teams managing microservices of the overall microservice system
Cooperating Teams (External Border):	All teams working on the same microservice system.
Managed Artifacts:	A subset of the microservices of the overall microservice system.
Date:	2020/06/11
Version of Document:	1.0

Comment:
This ßMACH protocol is based on a microservice definition, to test the ßMACH method and the microservice definition.

Figure 2. Context or meta-data of the definition of microservices in the ßMACH protocol. (See Fig. 3 and https://doi.org/10.5281/zenodo.10992169 for more details.)

	Product Knowledge	Demanded Knowledge	Roles	Process Knowledge	Process Information
Product Development	—	—	—	—	—
Product Maintains	—	—	—	—	—
Product Improvement	—	—	—	—	—
Explanation for Aspects the Team is Not Responsible for:					
Explanation for Aspects the Team is Responsible for:					
Inside Product Properties	Dependencies of microservices are minimised and do not need to be managed. [1]	No product knowledge to manage, so no demanded knowledge exists.	No product knowledge to manage, so no roles needed.	No product knowledge to manage, so no process needed.	No product knowledge to manage, so no information to collect.
Outside Product Properties	Defined by a subset of the system concerns (via the interfaces), managed by DevOps. [2]	The DevOps team has the knowledge how to do, the interface definition.	The DevOps team has to provide all needed roles. [3]	The DevOps team manages itself. [3]	Defined by a subset of the system concerns (via the interfaces), Managed by DevOps. [2]
Inside Dependencies	Dependencies of microservices are minimized and do not need to be managed. [1]	No product knowledge to manage, so no demanded knowledge exists.	No product knowledge to manage, so no roles needed.	No product knowledge to manage, so no process needed.	No product knowledge to manage, so no information to collect.
Outside Dependencies	Can be given by system concerns or by the DevOps team. [2]	No additional knowledge needed.	The DevOps team has to provide all needed roles. [3]	The DevOps team manages itself. [3]	Can be requested by the system concerns covered in this row. [2]
Inside Interfaces	Dependencies of microservices are minimized and do not need to be managed. [1] [2]	No product knowledge to manage, so no demanded knowledge exists.	No product knowledge to manage, so no roles needed.	No product knowledge to manage, so no process needed.	No product knowledge to manage, so no information to collect.
Outside Interfaces	Representation of a subset of the system concerns delivered to the enduser. [2]	Given by the outside responsibilities, managed by DevOps, so no open demands.	The DevOps team has to provide all needed roles. [3]	The DevOps team manages itself. [3]	Representation of a subset of the system concerns delivered to the enduser. [2]
Inside Responsibilities	Dependencies of microservices are mimized and do not need to be managed. [1]	No product knowledge to manage, so no demanded knowledge exists.	No product knowledge to manage, so no roles needed.	No product knowledge to manage, so no process needed.	No product knowledge to manage, so no information to collect.
Outside Responsibilities	Given by a subset of the system concerns. [2]	The separation of system concerns to microservices. [4]	??? [4]	??? [4]	Given by a subset of the system concerns. [2]
External Artifacts	Can be given by system concerns by the DevOps team. [2]	No additional knowledge needed.	The DevOps team has to provide all needed roles. [3]	The DevOps team manages itself. [3]	Can be requested by the system concerns covered in this row. [2]

Figure 3. Description of microservices based on the ßMACH protocol. ßMACH describes a set of key aspects. Each cell of the table represents an aspect. The right part of each cell holds the references to the definition in the ßMACH protocol (Fig. 1). ßMACH defines coloring. Based on the management process, we use light green in the right part of a cell for aspects that do not need active management. Active management means that an aspect is realized without a need for action. The darker green indicates that an aspect is also performed without needing active management but is provided by another. We use violet for aspects used or required by additional ones. Such an aspect indicates a special interest. Arrows with a peak-end describe a provides-relation. The aspect on the peak is provided by the other. A round end arrow gives a demand relation. The other aspect needs the one at the rounded end. The left part of the call can be colored, too. If the left part of multiple cells uses the same color, the cell's descriptions are equal or very similar. The described aspects in this figure are all based on the other parts of the ßMACH protocol, provided in Fig. 1 and 2. Download the ßMACH protocol as PDF to zoom in and explore the protocol (https://doi.org/10.5281/zenodo.10992169).

6. Results: learning from the ßMACH Protocol

We investigate the ßMACH protocol (Sec. 5.3):

1) We start to look for aspects that do not need active management. In the ßMACH protocol, such aspects are marked by a green color on the right part of the cell (Fig. 3). For the strict definition of microservices, the rows for inside aspects do not need active management. The reason is also present. Based on the independence from other services of the same system, no technical (product-based) cooperation with teams of the same microservice system is present. As a result, the other fields in the rows do not need active management because there is nothing to manage. There is no need to manage internal relations, a significant advantage. Fewer communication partners reduce the complexity and the needed team management skills. The team can concentrate on itself and is probably more productive. **ßMACH represents the idea of the strict microservice definition to foster scalability by separating microservices.**
2) Based on the provides and demands relations, the ßMACH protocol describes knowledge propagation. We already mentioned the propagation for the aspects without active management. The knowledge propagation for active management is interesting for a software engineer. How is the knowledge transferred and converted, and which knowledge is it? In Fig. 3, the starting point is the concerns. Individual microservices handle the system's business concerns (row outside responsibilities). The microservice team's responsibilities are the basis for the outside interfaces and the outside dependencies. Thus, the product properties are indirectly based on the concerns. In other words, the business concerns of the microservice need to be defined first. Afterward, the microservice team cares about creating and operating the microservice. **The team cares for the microservice. The ßMACH protocol points out that the team needs a defined business concern as a starting point and then manages the service creation and operation based on the concerns. Another influence on the team is not present.** (See Sec. 4 for other management decisions and strategies.)
3) Only one kind of description for roles is present in the ßMACH protocol (Fig. 3). The roles are not exactly defined. In other examples of ßMACH protocols, we have seen concrete roles like software developers, architects, and designers. In Fig. 3, there is a DevOps team. The roles this team needs are not fully defined. Based on the understanding of DevOps, the roles are reasonable to perform the given tasks. **The ßMACH protocol does not point out that the DevOps team needs to adapt to the microservice's business concern. For a concrete project, we need to define and instantiate the abstract definition of roles.**
4) The process knowledge is given by self-management of the DevOps team. Similar to the roles, this is not concrete. Nevertheless, DevOps is the idea of small teams and self-management. A concrete team should give more details on how

to perform self-management. **ßMACH points out the DevOps team's self-managed process. Thus, inadequate influences on the team have to be omitted. It is also a consequence of the independence of microservices.**

5) The definition of the management concept in ßMACH (Fig. 1) holds four easy-to-read bullet points. It is very minimal, easy to remember, fast to understand, and interpretation is present and referrable at any time (Fig. 3). Based on our observation, it is very helpful to have an explicit management process. It makes the process easier, reduces conflicts, and enables improvements. Also, a change in the process gets obvious, and changes can be explicitly discussed. **The ßMACH protocol is compact, and it is easy to understand the definition of the management process. So, the planned process is written down and can be referred to later on.** (See Sec. 4 for changing the management strategy and updating the protocol.)
6) The effort to create a ßMACH protocol is not very high and no special knowledge or skills are needed. To describe microservices, you need to understand microservices. So, you can create a protocol in about two hours on a whiteboard with the DevOps team. Afterward, the process is clear to all team members. We have also discussed two weeks about a single ßMACH protocol. We discussed the management process, and we learned a lot. Based on filling the ßMACH protocol, we identified the gaps in the process, found borderline cases, and nailed down the differences between our idea of the process and the practical doing. At least based on our observation, the ßMACH method supported us. **A ßMACH protocol is created in some hours and can help to improve the management process.**
7) ßMACH is a communication helper. The terms in the protocol support understandability. The description of the process by one person is easier to understand for the team. It was even possible to identify misunderstandings between persons. If two persons answered a management aspect in the ßMACH protocol differently, the process was not yet clear. **The definition of terms and the systematics of the ßMACH protocol support the communication of the involved persons and avoid misunderstandings in the management process.**
8) Our definition of microservices is strong for explaining the essentials of the concept. To allow minimal dependencies is a concession to practical implementations of microservice systems. Nevertheless, isolation is not easy to realize. Especially for transforming legacy systems to microservices, the definition gives a goal, not the transformation or an intermediate step. Thus, a practical realization of a microservice system probably sacrifices strict isolation and decides to manage the consequences instead of dealing with the realization of strict isolation. **We use a microservice definition to point out the advantages of strict isolation. A real-world system uses potbelly less strict definitions with reduced isolation. Especially, the transformation of a monolith into a**

microservice system will not hold our definition. In such a case, the ßMACH protocol will look different.

7. Conclusion

Based on performing the ßMACH method, we can state two kinds of findings.

First, the ßMACH method is helpful for the analysis of software management processes and supports the management. The method is systematic and defines terms to describe the process. So, it supports analyses, such as the understanding of the process by the development team and learning the process by all team members. Also, the method is easy and fast to perform and thus efficient.

Second, the isolation of individual microservices supports the development team. The team can avoid many aspects of management. In addition, the team can perform all the knowledge representation and transformation to develop an individual microservice. There are no supplementary relations or dependencies to the team. So, the number of teams can be scaled without overhead to individual teams.

We close this paper by introducing you to fill a ßMACH protocol for your software management process and take value from the method. To learn more about your management proceeding and how your colleagues understand it.

Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 414984028 – SFB 1404 FONDA

Appendix

ßMACH Protocol to Evaluate the Management of Software Artifacts

This is the ßMACH protocol. This protocol aims to improve the software artifact management. The target audience is persons in software development, software operation, software mentions, corresponding team leaders, managers of software artifacts, and those interested in improving the management of software artifacts on a small or big scale.
The method is to give the definition of software artifact management in this protocol and to give descriptions of different aspects of software management. To support all management aspects, you provide all descriptions (fill in all fields of the table below) based on your management definition. By giving the descriptions and extending the management definition until you have every description, you improve your management definition to respect all the different aspects of management given in this protocol. Your definition of the management process is not allowed to have parts not used by a description, So you avoid unnecessary parts of the definition.
You fill in this protocol before starting the software management process, ideally. So, you have the definition of the process present in time. During the management process, you can refer to it all the time. This protocol is an explicit description of your management concept. If something changes, you can create a new version of this protocol. From time to time, you check this protocol retrospective and identify which descriptions and parts of the definition are helpful and which aspects need improvement for the next version of this protocol or another protocol. So, you can improve your management skills.
The structure of the ßMACH protocol is as follows: A separate box asks for the context information. The protocol starts with the work package responsibilities to check your principal tasks. Following is the definition of your management concept and the table with the descriptions (for the different management aspects) you have to give. ßMACH recommends starting with the context, selecting the work packages, and then developing the definition while providing the descriptions stepwise.

Work Package Responsibilities:

Check this item if your team is responsible for product development. It means your team directly influences the creation of software artifacts, e.g., by programming modeling or supervising. Additionally, your team finishes the development.

☑ Finalizing Product Development

Check this item if your team is responsible for product maintenance. It means your team directly influences (non-functional) changes to the software artifacts while the artifacts are in operation, e.g., you are in a DevOps team or have to provide a bug fix. Additionally, the maintenance is not moved to another team later on.

☐ Finalizing Product Maintenance

Check this item if your team is responsible for product improvement. It means your team directly influences (functional) changes to software artifacts, e.g., you add additional features or develop an extended version. Additionally, no other team is improving the software artifacts later on.

☐ Finalizing Product Improvement

Definition of the Management Concept:

Here, you provide the definition of the software artifact management concept. You have to use each part of the definition in the description part below. Also, you have to base all descriptions (see below) on parts of this definition. The definition has to mark individual parts and give a reference for each part. You can use, e.g., an enumeration.
As a result, the definition is complete because you base all descriptions on the next page on the definition, and it is minimal because it does not have unused and unnecessary parts.

1. The software is created by a magical box.
2. The magical box works fully automatic.
3. The software is used exactly once, by the team itself.
4. Our team has to ask other teams to get the magical box and we have to return it.
5. The persons in our team can do tasks that not demand special knowledge or skills.
6. We do not want to learn from the project and we get a fixed price, so we deny documentation tasks.

Figure 4. Definition of software development toy example in the ßMACH protocol. The definition gives the numbers of the bullet points/parts. (See Fig. 6 and https://doi.org/10.5281/zenodo.10992007 for more details.)

Context (User/Team/Context Information):

Give the name of the person or persons that fill in this ßMACH protocol. Add company name, department, or whatever is needed to identify the person later.

Name of the Filler: Marcus Hilbrich

Give the team managing the software artifacts. It can be a single person managing some product artifacts, a five-person development team developing a single product component, or a whole organization managing multiple products. The definition of the team is essential because it gives the context for the descriptions in the later part and other questions. So, it highly influences the answers. You describe the team understandably, e.g., name all members or a generally known team name. In addition, the represented team defines the internals, and the cooperating teams define the external border.

Represented Team (Internal Border): Team 002

Give the teams you have to or can cooperate with. It can be teams of your organization you can emerge resources with, teams of other organizations that work on the same overall product, and teams you have to deliver or get something from (e.g., the billing team or a requirements engineering team). The set of teams defines what you can influence indirectly. The difference between cooperating teams and externals is that you cannot cooperate with externals (you have no direct influence on them). It is the outside border.

Cooperating Teams (External Border): All teams of the Magical Box Organisation

Give the artifacts to manage. It can be a set of concrete artifacts, a product, a project, or another understandable description. The description of the artifacts is essential because it describes what to manage, where there are possibly interfaces to artifacts that are not managed by your team or cooperating teams, etc.

Managed Artifacts: The boring game No. 3

Give the calendar date or timespan for filling this document. So, the coordination with other documents is possible.

Date: 04/29/2020

The version of the filled ßMACH protocol. The version number identifies minor or major changes and distinguishes documents.

Version of Document: 0.1.0 Initial Version

Here, you can put everything that has no other place in the document. It can be general remarks, changes to the last version, open questions, problems with the document itself, or whatever you can think of.

Comment:

This document is the description of Questionaries. All elements of the questionary are given in black, explanations to the elements are given in grey (next to the described element, and a very basic example of filling the ßMACH protocol is given in blue.)

Background colors in the following table are examples and not general, see the description of the table for future information.

Figure 5. Context or meta-data of software development toy example in the ßMACH protocol. (See Fig. 6 and https://doi.org/10.5281/zenodo.10992007 for more details.)

In the following table, you have to give the descriptions. You have to describe the management of different aspects based on the definition of the management concept (see above). The descriptions always concern the user/team/context information (see above). So, you always answer for your team and the artifacts you have given.
In case of multiple exemplifications that are very similar and based on the same part of the definition, give all corresponding fields of the table a unique text background color (different from white).
Right, in the field of an explanation, you have to give the set of parts of the definition of the management concept (an understandable reference) that you use for the corresponding description.
If nothing needs to be managed, based on the definition, mark the right part of the field (the references to the definition of the management concept) with a (light) green background color. An example is when you define the field as not relevant.
If the field is provided by another one and thus nothing needs to be managed, mark the right part of the field (the references to the definition of the management concept) with a dark green background color. An example is when a role provides knowledge, so the knowledge probably does not need individual management.
If the field is demanded by another one, mark the right part of the field (the references to the definition of the management concept) with a violet background color. (When the background is already green or light green, add a violate spot instead.) An example is when a role is needed to generate knowledge in a form defined by another field.

Figure 6. The text is part of the hints to the ßMACH protocol. It describes the filling in of the key aspects. The key aspects are separated into Fig. 7, 8, and 9. ßMACH describes a set of key aspects. Each cell of the table (Fig. 7, 8, and 9) represents an aspect. The right part of each cell holds the references to the definition in the ßMACH protocol (Fig. 4). ßMACH defines coloring. Based on the management process, we use light green in the right part of a cell for aspects that do not need active management. Active management means that an aspect is realized without a need for action. The darker green indicates that an aspect is also performed without needing active management but is provided by another. We use violet for aspects, used or required by additional ones. Such an aspect indicates a special interest. Arrows with a peak-end describe a provides-relation. The aspect on the peak is provided by the other. A round end arrow gives a demand relation. The other aspect needs the one at the rounded end. The left part of the call can be colored, too. If the left part of multiple cells uses the same color, the cell's descriptions are equal or very similar. The described aspects in this figure are all based on the other parts of the ßMACH protocol, provided in Fig. 4 and 5. Download the ßMACH protocol as PDF to zoom in and explore the protocol (https://doi.org/10.5281/zenodo.10992007).

	Product Knowledge		Demanded Knowledge		Roles		Process Knowledge		Process Information	
Explanation for Aspects the Team is Not Responsible for:										
Product Development Cross the fields in this row when your work package responsibility covers product development (check above).	If not crossed, give the information and knowledge your team has to provide to the development team(s). What are the demands for the software artifacts to build, what are the requirements, whom to contact, who has additional knowledge, who is responsible, who provides changes to consider, etc.? ———	–	If not crossed, give to be learned or not yet present knowledge that is important for the development team. Is domain knowledge required, are the software artifacts part of a specific process that has to be known, is knowledge about needed tools demanded, is knowledge for understanding provided documents needed, and similar? ———	–	If not crossed, give the required roles in your team to provide information and knowledge in this row, e.g., by teaching, learning, or using artifacts. ———	–	If not crossed, give the process your team uses to provide information and knowledge in this row. What is the (kind of) artifacts you create to provide information and knowledge? How are the artifacts created? What are the single steps, intermediated artifacts, the process, etc? ———	–	If not crossed, give information to collect during activities triggered by this row. Is it needed to write working hours, provide a list of invoked persons, have an access control list, is a protocol of tests required for certification, is a process documentation needed by an authority, or similar? ———	–
Product Maintenance Cross the fields in this row when your work package responsibility covers product maintenance (check above).	If not crossed, give the information and knowledge your team has to provide for the maintenance team. What is the software architecture, which compilers have to be used, where to find documentation, what are the requirements to fulfill, how to deploy the software, how to operate the software, etc? Based on 3, no maintains is needed.	3	If not crossed, give to be learned or not yet present knowledge that is important for the maintenance team. Is domain knowledge required? Are the software artifacts part of a specific process that has to be known? Is knowledge about used tools needed? Does the understanding of the provided documents require additional knowledge? Or other questions. Based on 3, no maintains is needed.	3	If not crossed, give the needed roles in your team to provide information and knowledge in this row, e.g., by teaching, learning, or using artifacts. Based on 3, no maintains is needed.	3	If not crossed, give the process your team uses to provide information and knowledge in this row. What is the (kind of) artifacts your team creates to provide information and knowledge? How are the artifacts created, what are the single steps, intermediated artifacts, the process, etc.? Based on 3, no maintains is needed.	3	If not crossed, give information collected during activities triggered by this row. Is it needed to write working hours, provide a list of invoked persons, have an access control list, is a test protocol required for certification, is a process documentation needed by an authority, or similar? Based on 3, no maintains is needed.	3
Product Improvement Cross the fields in this row if your work package responsibility covers product improvement (check above).	If not crossed, give the information and knowledge your team has to provide for the improvement team. What is the software architecture, which compilers have to be used, where to find documentation, what are the original requirements to fulfill, how to deploy the software, how to operate the software, etc? Based on 3, no improvement is needed.	3	If not crossed, give to be learned or not yet present knowledge that is important for the improvement team. Is domain knowledge required? Are the software artifacts part of a specific process that has to be known? Is knowledge about used tools needed? Does the understanding of the provided documents require additional knowledge? Or other questions. Based on 3, no improvement is needed.	3	If not crossed, give the roles in your team to provide information and knowledge in this row, e.g., by teaching, learning, or using artifacts. Based on 3, no maintains is needed.	3	If not crossed, give the process your team uses to provide information and knowledge in this row. What are the (kind of) artifacts your team creates to provide information and knowledge? How are the artifacts created, what are the single steps, intermediated artifacts, the process, etc.? Based on 3, no maintains is needed.	3	If not crossed, give information to collect during activities triggered by this row. Is it needed to write working hours, provide a list of invoked persons, have an access control list, is a test protocol required for certification, is a process documentation needed by an authority, or similar? Based on 3, no maintains is needed.	3

Figure 7. Description of software development toy example based on the ßMACH protocol. Presented is a set of key aspects. (See Fig. 6 and https://doi.org/10.5281/zenodo.10992007 for more details.)

Explanation for Aspects the Team is Responsible for:

	Product Knowledge		Demanded Knowledge		Roles		Process Knowledge		Process Information	
Inside Product Properties You have to give the management of the properties of the artifacts your team has to manage. It can be all kinds of requirements for the artifacts. In this row, you consider the product properties directly related to your team and demands by cooperating teams (related to the internal border).	Give the knowledge about the internal product properties. You describe the kind of used artifacts, the (kind of) knowledge about the product you need to manage, etc. Examples are requirement lists, descriptions of algorithms you implement, and domain knowledge. Based on 2 the box holdes the knowledge, a cooperation to other teams is not needed.	2	Give the knowledge your team needs (but does not yet have) to realize artifact management. Do you need knowledge, e.g., to use tools, communicate with other teams, or explore solutions? To get the box (4) or use it (2), no additional knowledge is needed.	2, 4	Give the roles you need in your team to handle product properties, e.g., by providing knowledge or performing tasks. To ask the other teams about the box does not demand a specific role.	4, 5	Give needed knowledge of your team to perform the process to realize the product properties. It can be (a part of) a software process model, a management concept, outsourcing, or whatever you want to do. Based on 1, 2 and 4 we have to get the magical box and start it, so no specific knowledge is needed (the person knows what to do).	1, 2, 4	Give the information that has to be collected while performing the management process in this row. Is it demanded to collect test results, do you have to give information for billing, is version management mandated, do you have to rate the used process, or something else? Not demanded by the definition.	6
Outside Product Properties You have to give the management of the properties of the artifacts your team (together with the cooperating teams) has to manage. It can be all kinds of requirements for the artifacts. In this row, you consider the product properties demanded by external parties (not by cooperating teams). It is related to the external border.	Give the knowledge about the external product properties. You describe the kind of artifacts to use, the (kind of) knowledge regarding the product you manage, etc. Examples are requirement lists, descriptions of the algorithm to implement, and domain knowledge. The box holdes the knowledge, the team does not managed it, additional knowledge is not needed.	2	Give the knowledge your team needs (but does not yet have) to realize artifact management. Does your team need the knowledge to use tools, communicate with other teams, explore solutions, etc.? To use the box (2), no additional knowledge is needed.	2	Give the roles in your team to handle product properties, e.g., by providing knowledge or performing tasks. To start the magical box does not demand a specific role.	2, 5	Give the knowledge needed to perform the process that realizes the product properties. It can be (a part of) a software process model, a management concept, outsourcing, or whatever you want to do. Based on 2, no noticeable process is needed.	2	Give the information to collect while performing the management process in this row. Is it demanded to collect test results, to give billing information, is version management mandated, do you have to rate the used process, or something else? Not demanded by the definition.	6
Inside Dependencies You have to give the management of the dependencies your team has to manage. It can be all kinds of dependencies of/to the artifacts, dependencies of/to persons, dependencies to tooling, and similar. In this row, you consider dependencies to cooperating teams (related to the internal border).	Give the (kind of) knowledge and the representation of the knowledge needed for your artifacts that your team delivers or shares with cooperating teams and your team demands. It can be cooperative development of artifacts like libraries, the demand to get requirements for your artifacts, etc. The magical box holds the knowledge, we depend on the other teams to get the box.	2, 4	Give the (kind of) knowledge and the representation of the knowledge needed for your team to work but which is not (completely) present in your team. Is it planned to teach your team the usage of tools, is it demanded to transfer the domain knowledge to your team (by teaching or adding a member), etc? The description does not lists knowledge not yet part of the team that is needed.	2	Give the roles in your team that handle dependencies in this row. Give the role that has to learn knowledge or use a shared artifact. No knowledge to manage is given in this row, so no role is needed.	5	Give dependencies in executing the management process. Is the ongoing of your team managed by another team (e.g., the Scrum master or product owner is not part of your team)? No knowledge that is needed to processed is given in this row, so no management process is needed.		Give dependencies of process information in this row. For example, if your team collects information when other teams deliver knowledge or how much your team is billed by cooperating teams. Not demanded by the definition.	6
Outside Dependencies Give the management of the dependencies your team (together with the cooperating teams) has to manage. It can be all kinds of dependencies of/to the artifacts, of/to persons, to tooling, and similar. In this row, you have to consider the dependencies demanded by external (not by cooperating teams) parties (related to the external border). External demands are generally risky.	Give the (kind of) knowledge and the representation of the knowledge used for your artifacts if it is related to external parties and your team demands it. It can be algorithm descriptions or results of external projects. External means there is no cooperation or influence to the external parties. Implied by 3, no outside dependencies are needed, the other parts of the definition do not introduce additional dependencies.	3	Give the (kind of) knowledge and the representation of the knowledge needed for your team to work but is not (completely) present in your team. Is it needed to teach your team the usage of tools, is it required to transfer the domain knowledge to your team (by teaching), etc.? The description does not lists knowledge not yet part of the team that is needed.	1, 2, 3, 4, 5, 6	Give the roles in your team that handle dependencies. Give the role that has to learn knowledge or use an external artifact. No knowledge is given in this row, so no role is needed.		Give dependencies in executing the management process. Is the ongoing of your team managed by an external party (e.g., the product owner (SCRUM) is not part of your team)? Nothing to manage, so no process needed.		Give dependencies of process information in this row. For example, your team collects the information when external parties deliver knowledge or how much is billed by external parties. Not demanded by the definition.	6

Figure 8. Description of software development toy example based on the ßMACH protocol. Presented is a set of key aspects. (See Fig. 6 and https://doi.org/10.5281/zenodo.10992007 for more details.)

	Product Knowledge	#	Demanded Knowledge	#	Roles	#	Process Knowledge	#	Process Information	#
Inside Interfaces Give the management of the interfaces your team uses. It can be descriptions of already fixed interfaces, the (kind of) documents your team creates, or what you have to negotiate with cooperating teams (related to the internal border). Interfaces always mean interfaces of artifacts your team manages. Otherwise, it is a dependency.	Describe the realization of the interfaces (product representation). Describe what is known about the interfaces, which artifacts are related, how to give the technical specification, etc. Not mentioned in definition (the magical box makes everything).	2	Give the (kind of) knowledge not present in your team to manage the interfaces, e.g., if your team (or someone else) develops the interface definition in the management process. No interfaces, so no missing knowledge in the team.		Give the roles in your team that care about the interfaces. Give the role that has to learn knowledge or use an external document. No interfaces, so no roles needed.		Give the process to manage the interfaces given in this raw. Is it needed to negotiate, is it demanded to discover interfaces (missing documentation), how to realize the interfaces, how and when are interfaces defined or altered, how to perform testing, etc? No interfaces, so no process needed.		Give the information that has to be collected while performing the management process in this row. Is billing relevant information needed, is interface documentation (API description) recorded, is documenting different versions with time stamps mandated, etc? Not demanded by the definition.	6
Outside Interfaces Give the management of the interfaces your team uses with external parties (the real world). It can be descriptions of already fixed interfaces, the (kind of) documents you have to create later on, or what you have to negotiate with external parties (related to the external border). Examples are user interfaces, machine interfaces, and system interfaces. Interfaces mean always interfaces of artifacts your team has to manage. Otherwise, it is a dependency.	Describe the realization of the interfaces (product representation). Describe what is known about the interfaces, which artifacts are related, how to give the technical specification, etc. Not mentioned in definition (the magical box makes everything).	2	Give the (kind of) knowledge not present in your team to manage the interfaces, e.g., if the management process has to develop the interfaces. No interfaces, so no missing knowledge in the team.		Give the roles in your team that care about the interfaces in this row. No interfaces, so no roles needed.		Give the process to manage the interfaces in this row. Is it needed to negotiate, is it required to discover interfaces (missing documentation), how to realize the interfaces, how and when are interfaces defined or altered, etc? No interfaces, so no process needed.		Give the information to collect about the interfaces in this row. Is billing relevant information needed, is interface documentation (API description) demanded, is it mandated to document different versions with time stamps, etc.? Not demanded by the definition.	6
Inside Responsibilities What is your team related to cooperating teams responsible for? How is it possible to measure the success of your team? Which of the described requirements is related to your team? How are shared responsibilities managed, etc.?	Give the responsibilities related to the product, e.g., based on requirements. 3 successes that no responsebilities are given, the other parts of the definition do not introduce additional dependencies.	3	Give the responsibility for demanded knowledge, how to find out if your team has acquired it. The magical box (holding the knowledge) is shared with cooperating teams, so we have to return it.	4	Give the roles in your team that have to manage the measurement and achievement of the team's responsibilities to cooperating teams. To return the magical box does not demand a special role.	5	Give the process to manage the responsibilities in this row. Do you have regular meetings, plans in case of not achieving goals, etc.? Returning the magical box is a task that das not need management (the persons in the team know what to do).	5	Give the information to the responsibilities your team manages in this row. Is it needed to collect information about the process, is it mandated to notice the fulfillment of responsibilities, what kind of documents to use, etc.? Not demanded by the definition.	6
Outside Responsibilities What is your team related to external parties responsible for? How to measure the team's success (combined with the cooperating teams)? Which of the described requirements is related to your team? How are shared responsibilities managed, etc.?	Give the responsibilities related to the product, e.g., based on requirements. 3 successes that no outside responsibilities exist, the other parts of the definition do not introduce additional responsibilities.	3	Give the responsibility for demanded knowledge. How to find out if you have acquired it? The description does not lists knowledge not yet part of the team but is needed.	1, 2, 3, 4, 5, 6	Give the roles in your team that manage the measurement and achievement of the teams' responsibilities to external parties. No knowledge is given in this row, so no role is needed.		Give the process to manage the responsibilities in this row. Do you have regular meetings, plans in case of not achieving goals, etc.? Nothing to manage, so no process needed.		Give the information to the managed responsibilities in this row. Does your team need to collect information during the process? Is it mandated to notice the responsibility fulfillment? What kind of documents to use, etc.? Not demanded by the definition.	6
External Artifacts What kind of artifacts from an external party are used by your team? (Not external artifacts represented by product properties or dependencies.)	Give the external artifacts your artifacts demand. Are you using an external library, need an external service, is a component delivered by an external party, etc.? Give what your team needs to know about such artifacts. Not mentioned in definition, the magical box manages everything.	2	Give the knowledge about external artifacts not known by your team. Is an external party integrating an external artifact, does only an external party know what such an artifact does or how to use it, etc.? Not mentioned in definition, the magical box manages everything.	2	Give the roles in your team that handle the external artifacts. No external artifacts are given, so no role is needed.		Give the management process to handle external artifacts. No external artifacts are given, so no management process is needed.		Give the information to collect about external artifacts. Are you billed for using the artifacts, when are the artifacts delivered, when have you used an external service, how to collect this information, etc.? Not demanded by the definition.	6

Figure 9. Description of software development toy example based on the ßMACH protocol. Presented is a set of key aspects. (See Fig. 6 and https://doi.org/10.5281/zenodo.10992007 for more details.)

References

[1] 4Soft GmbH in Zusammenarbeit mit dem Informationstechnikzentrum Bund: *V-Modell XT Bund, Das Referenzmodell für Systementwicklungsprojekte in der Bundesverwaltung*, Informationstechnikzentrum Bund im Auftrag des Beauftragten der Bundesregierung für die Informationstechnik, version: 2.3 ed., 2019.

[2] Alagarasan V.: Seven Microservices Anti-patterns, 2015. www.infoq.com/articles/seven-uservices-antipatterns.

[3] Anderson D., Anderson L.: *Beyond Change Management: How to Achieve Breakthrough Results Through Conscious Change Leadership*, BusinessPro collection, Wiley, 2010. https://books.google.de/books?id=WbpH7p5qQ88C.

[4] Armstrong D.J.: The Quarks of Object-Oriented Development, *Communications of the ACM*, vol. 49(2), pp. 123–128, 2006. doi: 10.1145/1113034.1113040.

[5] Assadi A.: What are microservices?, 2016. https://www.ibm.com/blogs/cloud-computing/2016/05/04/what-are-microserverices/.

[6] Basili V.R.: *Software modeling and measurement: the Goal/Question/Metric paradigm*, Tech. rep., 1992.

[7] Bauer F.L., Bolliet L., Helms H.J.: Software Engineering. Garmisch, Germany, 1969. Report on a conference sponsored by the NATO SCIENCE COMMITTEE.

[8] Berges M.P.: *Object-Oriented Programming through the Lens of Computer Science Education*, Dissertation, Technische Universität München, München, 2015.

[9] Bernard P.: *Foundations of ITIL® 2011 Edition*, Van Haren, 2011.

[10] Bourque P., Fairley R.E., Society I.C.: *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, IEEE Computer Society Press, Washington, DC, USA, 3rd ed., 2014.

[11] Brewster E., Griffiths R., Lawes A., Sansbury J.: *IT service management: a guide for ITIL foundation exam candidates*, BCS, The Chartered Institute for IT, 2012.

[12] Bucchiarone A., Dragoni N., Dustdar S., Lago P., Mazzara M., Rivera V., Sadovykh A. (eds.): *Microservices – Science and Engineering*, Springer, Cham, 2020. doi: https://doi.org/10.1007/978-3-030-31646-4.

[13] Bungay S.: *The Art of Action: How Leaders Close the Gaps between Pans, Actions and Results*, Nicholas Brealey, 2010.

[14] Cater-Steel A., Tan W.G.: *itSMF Australia 2005 Conference: Summary of ITIL adoption survey responses*, Ph.D. thesis, University of Southern Queensland, 2005.

[15] Cechini F., Ice R., Binkley D.: *Systems Engineering Guidebook for Intelligent Transportation Systems*, Tech. Rep. Version 3.0, U.S. Department of Transportation, Federal Highway Administration, California Division, 2009.

[16] Clarke P., Mesquida A.L., Ekert D., Ekstrom J.J., Gornostaja T., Jovanovic M., Johansen J., *et al.*: An Investigation of Software Development Process Terminology. In: P.M. Clarke, R.V. O'Connor, T. Rout, A. Dorling (eds.), *Software Process Improvement and Capability Determination*, pp. 351–361, Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-38980-6_25.

[17] Copei S., Zündorf A.: How to Synchronize Microservices. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_16.pdf.

[18] Crnkovic I., Sentilles S., Vulgarakis A., Chaudron M.R.V.: A Classification Framework for Software Component Models, *IEEE Transactions on Software Engineering*, vol. 37(5), pp. 593–615, 2011. doi: 10.1109/tse.2010.83.

[19] C/S2ESC – Software & Systems Engineering Standards Committee: IEEE 1074-2006 – IEEE Standard for Developing a Software Project Life Cycle Process, 2006.

[20] Dennis A., Wixom B., Tegarden D.: *Systems Analysis and Design with UML*, Wiley, 2009.

[21] Dragoni N., Giallorenzo S., Lafuente A.L., Mazzara M., Montesi F., Mustafin R., Safina L.: *Microservices: Yesterday, Today, and Tomorrow*, pp. 195–216, Springer International Publishing, Cham, 2017. doi: 10.1007/978-3-319-67425-4_12.

[22] Duell M., Goodsen J., Rising L.: Examples to Accompany: Design Patterns; Elements of Reusable Object-Oriented Software, 1997. [Online; accessed 31-Juli-2020].

[23] Dyck A., Penners R., Lichter H.: Towards Definitions for Release Engineering and DevOps. In: *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, pp. 3–3, 2015. doi: 10.1109/RELENG.2015.10.

[24] Edwards D.W.: *Out of the Crisis*, 1986. doi: 10.7551/mitpress/11457.001.0001.

[25] Elliott G.: *Global Business Information Technology: An Integrated Systems Approach*, Pearson Addison Wesley, 2004. https://books.google.ru/books?id=qGfzMlfgzEcC.

[26] Everett G.D., Raymond McLeod J.: *Software Testing; Testing Across the Entire Software Development Life Cycle*, John Wiley & Sons, Ltd, 2006. doi: 10.1002/9780470146354.fmatter.

[27] Fernández D.M., Wagner S.: Naming the Pain in Requirements Engineering: Design of a Global Family of Surveys and First Results from Germany, *CoRR*, vol. abs/1611.049761611.04976, 2016. http://arxiv.org/abs/1611.04976. 1611.04976.

[28] Fowler M.: *Domain-specific languages*, Pearson Education, 2010.

[29] Fowler M., Rice D.: *Patterns of Enterprise Application Architecture*, A Martin Fowler signature book, Addison-Wesley, 2003.

[30] Fritzsch J., Bogner J., Wagner S., Zimmermann A.: Microservices in the German Industry: Insights into Technologies, Characteristics, and Software Quality. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_25.pdf.

[31] Gabbrielli M., Lanese I., Zingaro S.P.: Microservice-Oriented Computing for the Internet of Things. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_3.pdf.

[32] Ginger L.: Embrace and Exploit Change as a Program Manager: Guidelines for Success, *Journal of Change Management*, vol. 10, pp. 2–13, 2012.

[33] Hasselbring W.: *Software Architecture: Past, Present, Future*, pp. 169–184, Springer International Publishing, Cham, 2018. doi: 10.1007/978-3-319-73897-0_10.

[34] Hausotter A., Koschel A., Lange M.: Microservices in Higher Education - Migrating a Legacy Insurance Core Application. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_8.pdf.

[35] Hiatt J.: *ADKAR: a model for change in business, government, and our community*, Prosci, 2006.

[36] Hilbrich M., Bountris V.: Are Workflows a Language to Solve Software Management Challenges?–A ßMACH Based Analysis. In: *New Trends in Intelligent Software Methodologies, Tools and Techniques*, Frontiers in Artificial Intelligence and Applications, vol. 355, pp. 221–232, IOS Press, 2022. doi: 10.3233/FAIA220253.

[37] Hilbrich M., Frank M.: Abstract Fog in the Bottle - Trends of Computing in History and Future. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 519–522, 2018. doi: 10.1109/SEAA.2018.00089.

[38] Hilbrich M., Jakobs C., Werner M.: Do Microservices Prevent High Qualitative Code? In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://microservices.fh-dortmund.de/papers/Microservices_2019_paper_9.pdf.

[39] Hilbrich M., Lehmann F.: Discussing Microservices: Definitions, Pitfalls, and their Relations. In: *2022 IEEE International Conference on Services Computing (SCC)*, pp. 39–44, IEEE Computer Society, Los Alamitos, CA, USA, 2022. doi: 10.1109/SCC55611.2022.00019.

[40] Hilbrich M., Lehmann F.: ßMACH – A Software Management Guidance. In: D.G. Reichelt, R. Müller, S. Becker, W. Hasselbring, A. van Hoorn, S. Kounev, A. Koziolek, R. Reussner (eds.), *Symposium on Software Performance 2021*, CEUR-WS, 2022. http://ceur-ws.org/Vol-3043/.

[41] Hui A.: Lean Change: Enabling Agile Transformation through Lean Startup, Kotter and Kanban: An Experience Report. In: *2013 Agile Conference*, pp. 169–174, 2013. doi: 10.1109/agile.2013.22.

[42] ISO/IEC JTC 1/SC 7 Software and systems engineering: ISO/IEC 15504-1:2004 Information technology – Process assessment – Part 1: Concepts and vocabulary, 2004.

[43] ISO/IEC JTC 1/SC 7 Software and systems engineering: ISO/IEC TS 24748-1:2016 Systems and software engineering - Life cycle management - Part 1: Guidelines for life cycle management, 2016.

[44] ISO/IEC JTC 1/SC 7 Software and systems engineering: ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software life cycle processes, 2017.

[45] Jabbari R., bin Ali N., Petersen K., Tanveer B.: What is DevOps? A Systematic Mapping Study on Definitions and Practices. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2962695.2962707.

[46] Johnson P., Ekstedt M., Jacobson I.: Where's the Theory for Software Engineering?, *IEEE Softw*, vol. 29(5), p. 96, 2012. doi: 10.1109/MS.2012.127.

[47] Kroll P., Kruchten P.: *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, no. 1 In Object Technology Series, Addison-Wesley Professional, 2003.

[48] Lau K., Wang Z.: Software Component Models, *IEEE Transactions on Software Engineering*, vol. 33(10), pp. 709–724, 2007. doi: 10.1109/tse.2007.70726.

[49] Lea G.: Why "Don't Use Shared Libraries in Microservices" is Bad Advice, 2016. http://www.grahamlea.com/2016/04/shared-libraries-in-microservices-bad-advice/.

[50] Lewis J., Fowler M.: Microservices: a Definition of this new Architectural Term, 2014. http://martinfowler.com/articles/microservices.html.

[51] Linthicum D.: Chapter 1: Service Oriented Architecture (SOA), https://web.archive.org/web/20160206132542 /https://msdn.microsoft.com/en-us/library/bb833022.aspx#, 2016. [Online; accessed 18-Juni-2020].

[52] Lu N., Glatz G., Peuser D.: Moving mountains – practical approaches for moving monolithic applications to Microservices. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_30.pdf.

[53] Marrone M., Kolbe M.: Impact of IT Service Management Frameworks on the IT Organization: An Empirical Study on Benefits, *Challenges, and Processes*, pp. 501–525, 2011.

[54] Martin R.C.: *Clean Code: A Handbook of Agile Software Craftsmanship*, Robert C. Martin Series, Prentice Hall, Upper Saddle River, NJ, 2008. https://www.safaribooksonline.com/library/view/clean-code/9780136083238/.

[55] Maschio B.: Updating the current Jolie microservices based DMS solution to include electronic invoicing. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_15.pdf.

[56] Mauro T.: Adopting Microservices at Netflix: Lessons for Architectural Design, 2015. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

[57] Microsoft: What are microservices?, 2022. https://azure.microsoft.com/en-us/solutions/microservice-applications/. [Online; accessed August-2020].

[58] Microsoft: Gründe für einen Microservice-Ansatz zum Erstellen von Anwendungen, Version: Jun 14, 2019. https://docs.microsoft.com/de-de/azure/service-fabric/service-fabric-overview-microservices.

[59] Nadareishvili I., Mitra R., McLarty M., Amundsen M.: *Microservice Architecture : Aligning principles, practices, and culture*, O'Reilly Media, 2016.

[60] Nash J., Ehrenfeld J.: Code Green: Business Adopts Voluntary Environmental Standards, *Environment: Science and Policy for Sustainable Development*, vol. 38https://doi.org/10.1080/00139157.1996.9930973(1), pp. 16–45, 1996. doi: 10.1080/00139157.1996.9930973. https://doi.org/10.1080/00139157.1996.9930973.

[61] Newman S.: *Building Microservices*, O'Reilly Media, 2015.

[62] Object Management Group (OMG): Automated Function Points (AFP), https://www.omg.org/spec/AFP/1.0/PDF, 2014. [Online; accessed 17-August-2020].

[63] Popper K.R.: *Objective Knowledge: An Evolutionary Approach*, Oxford, England: Oxford University Press, 1972. doi: 10.2307/2106696.

[64] Popper K.R., Eccles J.C.: *The Self and its Brain: An Argument for Interactionism*, Springer, 1977. doi: 10.2307/1577999.

[65] Pratt M.: Microservice Pitfalls & AntiPatterns, Part 1, 2016. https://homeadvisor.tech/software-antipatterns-microservices/.

[66] Ralph P.: Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering, *IEEE Transactions on Software Engineering*, vol. 45(7), pp. 712–735, 2019. doi: 10.1109/TSE.2018.2796554.

[67] Rational Software: Rational Unified Process, Best Practices for Software Development Teams, https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf, 1998. [Online; accessed 18-Juni-2020].

[68] Richards M.: *Microservices Antipatterns and Pitfalls*, O'Reilly Media, 2016.

[69] Scacchi W.: Process models in software engineering, *Encyclopedia of software engineering*, 2001. doi: 10.1002/0471028959.sof250.

[70] Schwaber K., Sutherland J.: *The Scrum Guide^{TM}, The Definitive Guide to Scrum: The Rules of the Game*, https://www.scrumguides.org/, 2017.

[71] Simon H.A.: *The Sciences of the Artificial*, MIT Press, 1996. doi: 10.7551/mitpress/12107.001.0001.

[72] Sjøberg D.I.K., Dybå T., Anda B.C.D., Hannay J.E.: *Building Theories in Software Engineering*, pp. 312–336, Springer London, London, 2008. doi: 10.1007/978-1-84800-044-5_12.

[73] Sneed H.M.: *Software Management*, Rudolf Müller online DV-Praxis, Köln, 1987.

[74] Sommerville I.: *Software Engineering*, Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England, tenth ed., 2016.

[75] Staples M.: Critical rationalism and engineering: ontology, *Synthese*, vol. 191(10), pp. 2255–2279, 2014. doi: 10.1007/s11229-014-0396-3.

[76] Staples M.: Critical Rationalism and Engineering: Methodology, *Synthese*, vol. 192(1), pp. 337–362, 2015. doi: 10.1007/s11229-014-0571-6.

[77] Stein A., Zillekens M., Khan M.: A Microservice architecture for monitoring, processing and predicting climate data in animal husbandry. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_28.pdf.

[78] Steinacker G.: Why Microservices?, 2016. https://www.otto.de/jobs/technology/techblog/artikel/why-microservices_2016-03-20.php. [Online; accessed August-2020].

[79] Szyperski C., Gruntz D., Murer S.: *Component Software: Beyond Object-Oriented Programming*, ACM Press and Addison-Wesley, 2nd ed., 2002.

[80] The Standish Group International, Inc.: *The CHAOS Report (1994)*, Tech. rep., 1994.

[81] The Standish Group International, Inc.: *Chaos Report 2015*, Tech. rep., 2015.

[82] Thompson M.: Why SaaS and Microservices are Critical to Developing in the Cloud, 2015. https://www.rightbrainnetworks.com/2015/01/29/why-saas-and-microservices-are-critical-to-developing-in-the-cloud/.

[83] Tilkov S.: Microservices: A Taxonomy. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_29.pdf.

[84] Tracz W.: DSSA (Domain-Specific Software Architecture): Pedagogical Example, *SIGSOFT Softw Eng Notes*, vol. 20(3), pp. 49–62, 1995. doi: 10.1145/219308.219318.

[85] Ullenboom C.: *Java ist auch eine Insel*, Galileo Computing, Bonn, 6., aktualisierte und erweiterte Auflage ed., 2007. http://www.galileocomputing.de/openbook/javainsel6/.

[86] U.S. Environmental Protection Agency, Office of Atmospheric Programs, Climate Protection Partnerships Division: Clean Energy-Environment Guide to Action, Policies, Best Practices, and Action Steps for States, https://web.archive.org/web/20120713125427/http://www.epa.gov/statelocalclimate/documents/pdf/guide_action_full.pdf, 2006. [Online; accessed 13-August-2020].

[87] Vasanthapriyan S., Tian J., Xiang J.: A Survey on Knowledge Management in Software Engineering. In: *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, pp. 237–244, 2015. doi: 10.1109/QRS-C.2015.48.

[88] Vega N.: Answering Your Microservices Webinar Questions, 2015. https://www.ibm.com/blogs/bluemix/2015/02/answering-microservices-webinar-questions/#q1.

[89] Wetherill J.: Microservices and PaaS (Part I), 2014. https://dzone.com/articles/microservices-and-paas-part-1.

[90] Wilde N., Gonen B., El-Sheikh E., Zimmermann A.: *Approaches to the Evolution of SOA Systems*, pp. 5–21, Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-40564-3_2.

[91] Wohlin C., Šmite D., Moe N.B.: A general theory of software engineering: Balancing human, social and organizational capitals, *Journal of Systems and Software*, vol. 109, pp. 229–242, 2015. doi: 10.1016/j.jss.2015.08.009.

[92] Wolff E.: *Microservices: Grundlagen flexibler Softwarearchitekturen*, dpunkt.verlag GmbH, 2015.

[93] Wolff E.: Why Microservices Fail: An Experience Report. In: *International Conference on Microservices*, University of Applied Sciences and Arts Dortmund, Germany, 2019. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_18.pdf.

Affiliations

Marcus Hilbrich
Humboldt-Universität zu Berlin, Department of Computer Science, Berlin, Germany, marcus.hilbrich@informatik.hu-berlin.de

Ninon De Mecquenem
Humboldt-Universität zu Berlin, Department of Computer Science, Berlin, Germany, mecquenn@informatik.hu-berlin.de

Received: 05.05.2023
Revised: 10.04.2024
Accepted: 15.04.2024