Microservices Tenets: Agile Approach to Service Development and Deployment

Overview and Vision Paper, SummerSoC 2016

Olaf Zimmermann[1]
[1] University of Applied Sciences of Eastern Switzerland (HSR FHO),
Oberseestrasse 10, 8640 Rapperswil, Switzerland
ozimmerm@hsr.ch

Abstract. Some microservices proponents claim that microservices form a new architectural style; in contrast, advocates of Service-Oriented Architecture (SOA) argue that microservices merely are an implementation approach to SOA. This overview and vision paper first reviews popular introductions to microservices to identify microservices tenets. It then compares two microservices definitions and contrasts them with SOA principles and patterns. This analysis confirms that microservices indeed can be seen as a development- and deployment-level variant of SOA; such microservices implementations have the potential to overcome the deficiencies of earlier approaches to SOA realizations by employing modern software engineering paradigms and Web technologies such as domain-driven design, RESTful HTTP, IDEAL cloud application architectures, polyglot persistence, lightweight containers, a continuous DevOps approach to service delivery, and comprehensive but lean fault management. However, these paradigms and technologies also cause a number of additional design choices to be made and create new options for many "distribution classics" type of architectural decisions. As a result, the cognitive load for (micro-)services architects increases, as well as the design, testing and maintenance efforts that are required to benefit from an adoption of microservices. To initiate and frame the buildup of architectural knowledge supporting microservices projects, this paper compiles related practitioner questions; it also derives research topics from these questions. The paper concludes with a summarizing position statement: microservices constitute one particular implementation approach to SOA (service development and deployment).

Keywords: architectural principles, architectural styles, domain-driven design, IDEAL cloud application architectures, DevOps, loose coupling, messaging, patterns, REST, service-oriented computing, SOA, systems management

1. Introduction and Position Overview

No consensus regarding the relationship between Service-Oriented Architecture (SOA) and microservices has been reached so far. This paper argues that microservices concepts and technologies do not constitute a new architectural style different from SOA, but rather qualify as SOA implemented and services realized in one particular way with state-of-the-art software engineering practices. This position is derived from a literature review. This review started with the reading list and the outcomes of a microservices workshop at the SATURN 2015 practitioner conference [4][1]. It analyzed

[1] This workshop was organized and primarily attended by practicing architects and thought leaders (rather than service-oriented computing researchers or microservices advocates).

introductory articles that characterize microservices, industrial case studies, and emerging microservices patterns[2].

The literature review makes evident that the differences between microservices and previous attempts to service-oriented computing do not concern the *architectural style* as such (i.e., its design intent/constraints and its platform-independent principles and patterns [33]), but its concrete *realization* (e.g., development/deployment paradigms and technologies). For instance, the logical application and integration designs of many microservices architects are geared towards continuous delivery and hosting services in cloud computing offerings, and they decide for Web-centric technology stacks and/or pre-packaged open source assets such as MongoDB, Express, AngularJS and Node.js, sometimes abbreviated MEAN [21]. These choices do not violate any SOA principles or patterns such as *loose coupling* and *service contract* [8,13,33], but rather embrace and leverage them.

The literature review also unveils that, just like any incarnation of SOA, microservices architectures are confronted with a number of nontrivial design challenges that are intrinsic to any distributed system – including data integrity and consistency management, service interface design and evolution, and application/service management (including application and infrastructure security management); such architecture design issues transcend both style and technology debates [4].

The paper presents these positions in the following steps: first trend topics from the microservices literature are collected and distilled into seven microservices tenets. Two popular microservices definitions are then compared by viewpoint and design intent and analyzed with respect to their SOA positions (Sections 2 and 3). Section 4 highlights critical gaps in the microservices literature in the form of practitioner questions. These questions are then grouped and generalized to identify research areas and related problems/questions. Section 5 summarizes and concludes.

2. Microservices Trend(s) in Industry and Academia

State of the practice. In recent years, a shift of focus in developer communities and publications could be observed: from people and processes (e.g., agile practices such as user storytelling and test automation) to integration technology and application hosting (e.g., RESTful HTTP, cloud computing, DevOps). Under the umbrella term *microservices*, renewed interest in architecture and design can be observed at present (similar in intensity to the early days of the patterns movement). Discussing quality attributes such as scalability and performance or choosing patterns such as "service contract" [33] or "API gateway" [25] no longer seems to indicate violations of the "you aren't gonna need it" (YAGNI) principle or a suffering the "big design upfront" (BDUF) fallacy; project team members are no longer considered to be "architecture astronauts" [27] when considering and arguing about microservices *architectures*. Agile architecture represents a consensus position between process and structure [15].

According to case studies in the literature, e.g., [2,7,28], successful microservices architecture designs and microservices deployments are made possible by modern

[2] Articles that are rooted in actual project experience, but not peer-reviewed and published in academic venues were considered to be relevant and eligible for the literature review.

software engineering paradigms and recent advances in Web application development – for instance, a) domain-driven design and test-driven development, b) IDEAL pipes-and-filters chaining of fine-grained processing logic, c) polyglot programming and persistence, d) build and test process automation and continuous deployment, e.g., into lightweight containers and cloud computing environments and e) lean approaches to systems management closely intertwined with software construction (alias DevOps [11]). The literature also points out that there is no "one-size-fits-all": microservices are not always suited as SOA implementation approach as certain prerequisites in the (business) problems to be solved and the project context have to be met [4,5].

State of the art (in academia). The term microservices originates from agile developer communities and has appeared in blog posts and online articles since 2014; see [17] for a brief anthology. Academic conferences that focus on services, or at least refer to service-oriented computing in their calls for papers as one of several topic areas, are only beginning to pick up the microservices trend/topic, e.g., in keynotes and workshops, e.g., at ICWE 2016 and ESOCC 2016. At the time of writing, very few (if any) peer-reviewed research papers on microservices existed.

3 Microservices Tenets vs. SOA Principles and Patterns

This section first identifies the common elements in popular microservices definitions and contrasts the differing SOA positions in two of these definitions with each other. It then delivers a detailed comparison based on seven tenets and 4+1 *viewpoints* [16].

The following common themes recur in the introductory literature and case studies on microservices [2,7,17,19,22,23,25]:

1. *Fine-grained interfaces* to single-responsibility units that encapsulate data and processing logic are exposed remotely, typically via RESTful HTTP resources or asynchronous message queues. These remote units constitute services that can be deployed, changed, substituted, and scaled independently of each other.
2. Business-driven development practices and pattern languages such as *Domain-Driven Design (DDD)* [3] are employed to identify and conceptualize services.
3. Cloud-native application design principles are followed, e.g., as summarized in *IDEAL* (Isolated State, Distribution, Elasticity, Automated Management and Loose Coupling) [8] or the twelve app factors in Heroku's method [32].
4. Multiple computing paradigms (such as functional and imperative) and storage paradigms are leveraged (e.g., relational databases and several types of NoSQL stores) in a *polyglot programming and persistence* strategy. Some of these polyglot services only guarantee eventual rather than strong consistency.
5. *Lightweight containers* are used to deploy services. Docker and Dropwizard are frequently mentioned as two related options (although these two technologies do not reside on the same level of abstraction and have rather different scopes, operating system virtualization vs. code library assembly).
6. *Decentralized continuous delivery* is practiced during service development (which requires/promotes a high degree of automation and autonomy).

7. *DevOps*: Lean, but holistic and largely automated approaches to configuration, performance and fault management are employed, which extend agile practices and include service monitoring.

With respect to SOA, the following two contrary positions define the respective ends of the spectrum:[3]

- *Microservices as a new architectural style* that can be contrasted against SOA (which also is positioned as an architectural style [33]): "The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies" (J. Lewis and M. Fowler [17]). Detailed explanations and examples of nine characteristics derived from this rather dense definition can be found in [17].
- *Microservices as one way of doing SOA (right)*: "The microservices approach has emerged from real-world use, taking our better understanding of systems and architecture to do SOA well. So you should instead think of microservices as a specific approach for SOA in the same way that XP or Scrum are specific approaches for Agile software development." (S. Newman [23]).

  Newman moves on to define microservices via the following principles [23]:

  1. "Model around business concepts", to be represented as bounded contexts and domain models according to Domain-Driven Design (DDD) patterns [3].
  2. "Adopt a culture of automation" in testing and deployment; practice continuous delivery.
  3. "Hide internal implementation details" such as databases; define technology-agnostic Application Programming Interfaces (APIs).
  4. "Decentralize all the things": e.g., apply shared governance, prefer service choreography over orchestration, use dumb middleware but smart endpoints.
  5. Make services "independently deployable", e.g., let versioned (service) endpoints co-exist; deploy only one service per (virtual) host.
  6. "Isolate failure", e.g. introduce circuit breakers to make services robust.
  7. Be "highly observable", e.g. via semantic monitoring with data aggregation.

While the definition by Lewis and Fowler contains nine characteristics, Newman establishes seven principles. They overlap, but also differ substantially (Table 1).

[3] The rationale for the selection of these two particular sources is a) the generality and breadth of the discussions and b) their popularity.

Table 1. Comparison of definitions (with principle-to-characteristics mapping).

Characteristics by Lewis/Fowler [16]	**Relationship**	**Newman's Principles [23]**
1. Componentization via services (running in own process and communicating with lightweight mechanisms)	(similar to)	Hide internal implementation details
2. Organized around business capabilities	(matches)	Model around business concepts
3. Products not projects	(no pendant)	
4. Smart endpoints and dumb pipes	(included in)	Decentralize all the things
5. Decentralized governance (enabling polyglot programming)	(superset of)	
6. Decentralized data management (and polyglot persistence)	(superset of)	
7. Infrastructure automation (and decentralized management)	(superset of)	Adopt a culture of automation
(attribute in definition, but not elaborated upon in dedicated section of article)	(matches)	Independently deployable
8. Design for failure	(subset of)	Isolate failure
9. Evolutionary design	(no pendant)	
	(no pendant)	Highly observable

Several other introductions exist, which list similar microservices tenets [2,19,25]. The microservices movement has received a lot of attention in online publications; many reactions have been positive, but sceptic ones can be found as well [18,26,29,31], e.g., "microservices is SOA, for those who know what SOA is" [12].

SOA vs. microservices. Let us now map the defining elements in the two above definitions to SOA principles and patterns as defined in the academic literature, including our own work [33], but also practitioner articles and books such as [9,13]. Table 2 analyzes the definition from [17] to identify SOA pendants in the literature.

Table 2. Analysis of characteristics of microservices in definition from Lewis and Fowler.

Microservices	**Viewpoint and Quality Intent**	**SOA Pendant**
Componentization via services	Logical/Process Viewpoint (VP) [16]: separation of concerns improves modifiability and scalability	Service provider, consumer, contract (same concept) [6,9,33]
Organized around business capabilities	Scenario VP: domain model and ubiquitous language [3] make code understandable and easy to maintain	Key part of SOA definitions in books and articles since 2003 [6,9,33]
Products not projects	n/a (not technical but process-related)	Enterprise SOA programs often organized by service products [35]
Smart endpoints and dumb pipes	Process VP (related to integration): information hiding improves scalability and modifiability	Same best practice design rule exists for SOA, e.g., Enterprise Service Bus (ESB) design/usage; risk of misuse presumably higher in SOA (time will tell for microservices) [12]
Decentralized governance	n/a (not technical but process-related)	SOA governance (might be more centralized, but does not have to) [9]

Decentralized Data Management (DM)	e.g. Logical VP, Physical VP: polyglot persistence promotes flexibility and suitability	Same (de)centralization options; NoSQL more recent than SOA
Infrastructure automation	Development VP, Physical VP: speed, repeatability	No direct pendant (not style-specific, more recently advanced)
Design for failure	All VPs: robustness, reliability	Key concern for any distributed system, SOA or other
Evolutionary design	n/a (not technical but process-related): improves replacability, upgradeability	Service design methods, backward compatible contracts

The table unveils that several of the nine characteristics of microservices (e.g., "products not projects") primarily pertain to the development process/culture and the process and physical viewpoints, *not* the logical architectural patterns used. Most characteristics do have SOA pendants. Indeed, many existing SOA patterns and best practices can be found in the microservices literature (often under different names, e.g. the "API Gateway" [25] shares some intent, responsibilities and underlying atomic patterns with SOA-style ESBs, e.g., Message Transformation [10] and Remote Façade [6]). Decentralization is emphasized more than earlier SOA literature did.

An inspection of the second definition [23] yields similar results (Table 3).

Table 3. Analysis of Newman's principles of microservices.

Microservices	**Viewpoint, Intent**	**SOA Pendant**
Model around business concepts	Scenario Viewpoint (VP), intent: see Table 1	Key part of most SOA definitions since 2003, see e.g. Chapter 2 in [33]
Adopt a culture of automation	Process VP, Physical VP, intent: see Table 1	No direct SOA pendant (see Table 1)
Hide internal implementation details	Logical VP, Development VP: flexibility, portability, maintainability	Important architectural principle and development idiom (common sense) irrespective of style (but promoted by most styles) [6,9,33]
Decentralize all the things	n/a (not technical but process-related)	SOA governance, might be more centralized, but does not have to (see Table 1) [9]
Independently deployable	Process VP: frequent releases/incremental updates, scalability	No direct pendant in style, but precursor attempts such as Service Component Architecture (SCA) [24], an OASIS specification with vendor and open source implementations
Isolate failure	All VPs, intent: robustness (see Table 1)	Done in any distributed computing approach (hopefully)
Highly observable	Process VP, Physical VP: manageability, maintainability	Done in any distributed computing approach (hopefully)

None of Newman's seven principles focusses on the logical viewpoint exclusively; some of them are not specific to microservices, but represent good advice to designers of any distributed system (e.g., "isolate failure"). Newman's position that microservices form one specific approach to SOA w.r.t. development and deployment (but also project organization and engineering process) is backed by the table data. Figure 1 summarizes the analysis by positioning the seven tenets established at the beginning of this section (T-x), the nine characteristics from Lewis and Fowler (LF-y), and Newman's seven principles (N-z) in Kruchten's 4+1 viewpoint scheme [16].[4]

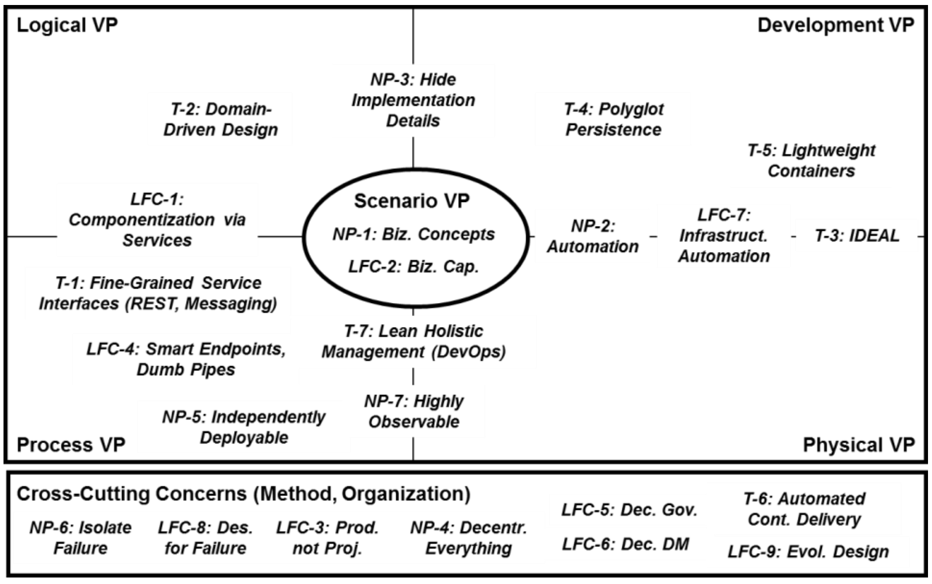
Logical VP
Development VP
NP-3: Hide Implementation Details
T-2: Domain-Driven Design
T-4: Polyglot Persistence
T-5: Lightweight Containers
LFC-1: Componentization via Services
Scenario VP
NP-1: Biz. Concepts
LFC-2: Biz. Cap.
NP-2: Automation
LFC-7: Infrastruct. Automation
T-3: IDEAL
T-1: Fine-Grained Service Interfaces (REST, Messaging)
T-7: Lean Holistic Management (DevOps)
LFC-4: Smart Endpoints, Dumb Pipes
NP-7: Highly Observable
NP-5: Independently Deployable
Process VP
Physical VP
Cross-Cutting Concerns (Method, Organization)
NP-6: Isolate Failure
LFC-8: Des. for Failure
LFC-3: Prod. not Proj.
NP-4: Decentr. Everything
LFC-5: Dec. Gov.
LFC-6: Dec. DM
T-6: Automated Cont. Delivery
LFC-9: Evol. Design

Fig. 1. Microservices tenets, characteristics, and principles by 4+1 viewpoints.

The figure shows consensus and/or complementary positions in three viewpoints (scenario, development, and process) and little focus on the remaining two (logical, physical); one tenet, five L/F characteristics and two N principles deal with cross-cutting concerns that span multiple viewpoints (e.g., decentralized governance).

Interpretation of analysis results and critique. Table 2, Table 3 as well as Figure 1 support Newman's evolutionary microservices vs. SOA position well; there is little (if any) evidence for the claim that microservices form their own novel style.

The order of Newman's principles seems to be more cohesive and easier to follow than the order of Lewis and Fowler's characteristics (e.g., less severe viewpoint switches occur when reading one-by-one). Regrettably, both definitions mix process, architecture, and development concerns (e.g., three of nine characteristics in [17] are related to the development process or deal with governance concerns). In software architecture research, architectural styles are typically defined via design intent, principles and patterns (like SOA [33]) or via (technical) constraints (like REST [1]); it is questionable whether process-related and organizational aspects should be included in such definitions. Their inclusion in [17] and [23] presumably is motivated by M. Conway's law, which states that designs mirror communication structures in organizations. However, such aspects do not allow architects to recognize the style (or variant of style) in the code easily, e.g., when reviewing actual architectures. This limits the usability of the definition (for instance, is "microservices project" an oxymoron?);

[4] Note that the scenario viewpoint has a retrospective role in [16], but is used differently here, representing the entire business perspective; the process viewpoint also covers integration and remoting concerns according to [16].

such hybrid approach also violates the separation-of-concerns and single-responsibility principles originally established for modules, but also applicable to definitions.

The provided process-related, organizational guidance does have value as it can be seen as an enabler and critical success factor. However, the relationship between an architectural style and an engineering process and culture can be characterized as "cross fertilization" or prerequisite (but not as inclusion); hence, this important information would be more consumable and easier to apply if it appeared in a separate, dedicated place (e.g., an enumeration or section devoted to these aspects).

4. Practitioner Questions and Research Topics/Problems

Even the proponents of microservices architectures (as a variant of/implementation approach to SOA) agree that getting microservices right is hard; microservices are not suited for each and every project/program/application landscape (in particular at the early evolution stages of such efforts [5]). Reasons include the inherent complexities and subtleties of distributed computing, but also the fine-grained, highly flexible and dynamic nature of microservices architectures that is emphasized in the seven microservices tenets from Section 3 (e.g., "independently deployable services"). Hence, design and decision guidance [34] continues to be highly desirable, or even becomes increasingly important (even if middleware and tools, e.g., those for continuous delivery and DevOps, promise to free architects, developer, and operators from having to design, implement, and perform many routine tasks).

Software architects and developers that consider adopting the microservices approach as their SOA implementation paradigm would like to learn from early adopters and thought leaders. Regrettably, the existing published architectural knowledge, e.g., draft pattern languages, is still rather vague on a number of design concerns, a.k.a. *architectural decisions required* [34]. This observation is supported by the following list of Practitioner Questions (PQs), some of which are backward looking and some of which are forward looking:[5]

1. Can you share any experiences and/or hint how to "sell" an investment in microservices to business stakeholders (e.g., project sponsors, product managers, C-level management in business and IT organizations)? How well do these experiences and hints relate to and/or align with the microservices tenets?
2. In which business domain and socio-technical context (or application genre and software operating range w.r.t. quality attributes [30]) have you applied microservices concepts and technologies – either successfully or unsuccessfully?
3. Which microservices principles (e.g., "independently deployable services" [23]) did you use in your microservices architecture designs, and how did you implement them (patterns, frameworks, middleware, tools)? Did you deploy to public or private cloud offerings (e.g. Amazon Lambdas, Google Cloud Function) or to more traditional application hosting environments?

[5] The above list is (roughly) ordered by phases in the software lifecycle; individual questions progress from abstract to concrete and from a more logical to a more physical view. The questions address practitioners with microservices experience ("you") so that they can be used in interviews and assessments, e.g. when evaluating offerings w.r.t. maturity and vision.

4. How do you see the relationship between REST and microservices? Is the usage of Web protocols required and sufficient? Or is RESTful HTTP only one of several valid remote communication options in the microservices architect's toolbox and, if so, what are the decision drivers when choosing an option?
5. How did you find an adequate/a suited service cut (e.g., how small/fine is small/fine enough)? How can Domain-Driven Design (DDD) [3] (and/or other approaches to application scoping and functional partitioning) be applied to decompose monoliths into services – practitioners have reported that they would welcome guidance that is more concrete than the rather frequently stated advice "define a bounded context for each domain concept to be exposed as service"?
6. How did you overcome "distribution classics" design challenges such as service lifecycle management, data representation/schema mismatches, service versioning and evolution (e.g., change of interface in terms of syntax and/or semantics), and error handling on your projects?
   - Did you define machine-readable service contracts? If so, what should they cover and how should they be expressed (e.g., are all REST maturity required, including support for Hypertext as the Engine of Application State (HATEOAS) [1], is billing information included)? If not, how did you achieve syntactic and semantic interoperability between service consumers and providers?
   - How did you deal with audit requirements, e.g., Completeness, Accuracy, and Validity of, as well as Restricted access to financially relevant business objects (e.g., CAVR controls) [14]?
   - Should overall, end-to-end data integrity be ensured in micro-services architectures, either centrally or de-centrally? If so, how to manage views, foreign key relationships and other semantic links across microservice boundaries? And how to backup an entire service landscape at once (atomic system snapshot, incremental backup)?
7. Do you have any advice/guidance how to compose microservices into end user client applications? How about application-level intermediaries, i.e., can micro-services also be clients of other microservices? If so, how to avoid microservice deployment dependency and dynamic invocation "spaghetti" (e.g., cycles, overly deep invocation chains)? Do you see a need for/can you recommend any tools, libraries, frameworks, middleware that can assist with this task, or is plain old development (applying state-of-the-art software engineering practices) sufficient?
8. Can you report on your technical and organizational scaling strategies (e.g., when having to deal with large services landscapes and rich/complex domain models with hundreds or thousands of interconnected entities)? Which tactics and patterns support these strategies well?
9. Which research and development challenges for a broad and sustainable adoption of microservices can the service-oriented computing community derive from your experience?

The above PQs result from the SOA/microservices literature review as well discussions with more than 10 industry thought leaders, enterprise application development and integration project practitioners, and SOA/service-oriented computing community members since early 2015. To compile it, we first reviewed microservices articles w.r.t. the tenets and resulting design challenges, then refined the findings in discussions with practicing architects and finally revisited older SOA literature including own (re-)collections of recurring architectural decisions and published experience reports. This three-step process was iterated through five times – reviewing, refactoring, and revising drafts of the questionnaire along the way.

The nine aggregated PQs make evident that many well-known distributed application/infrastructure architecture design challenges retain, and additional ones arise (due to the novel aspects/facets of microservices). Partial solutions exist in industry and academia; hence, a number of research topics can be derived from them.

Service interface design (contracting and versioning). HTTP goes a long way in standardizing a unified application-level communication interface (i.e., transfer protocol). However, the vast amount of HTML descriptions of Web APIs defined in Swagger or "Plain Old HTML" (POH) makes evident that not all interoperability concerns are covered by RESTful HTTP contracts (e.g., invocation semantics, message exchange formats, quality-of-service characteristics); dynamic service contracts and their auto-discovery at runtime are not always applicable, e.g., under audit requirements such as CAVR controls [14]. In general, syntactic and semantic contracts always exist, either implicitly or explicitly (as machine- and human-readable contracts). S. Allamaraju, a pragmatic rather than orthodox "RESTafarian", states that "distributed applications using HTTP as an application protocol, and built RESTfully, do have a contract, but of a different nature and kind" and "research and development opportunities abound" [1]. For instance, the role of Domain-Driven Design (DDD) in interface design has to be clarified and possibly supported by methods and tools.[6] Furthermore, backward compatibility has to be addressed, with "no versions at all" and "idempotency of services" being among the design options. An exact, formal definition of idempotency in this context is needed, as well as architectural patterns to design and test for idempotency in business object-centric enterprise applications, e.g., (information) systems of record and system of engagement. Moreover, RESTful HTTP is only one of several remoting options according to the microservices tenets established and definitions analyzed in Section 3, with messaging being an important alternative. If this "polyglot remoting" assumption holds, service contracts have to handle (at least) HTTP and the Advanced Message Queuing Protocol (AMQP). This research challenge originates from PQs 4, 5, and 6.

Microservice assembly and hosting. It is not fully understood yet how to create larger processing units (e.g., end-user applications) from a collection or repository of microservices.[7] It is also not clear whether there is a continuum from fine-grained microservices to coarser grained remote facades or to end user applications: Do both "macro" and "micro" services have their place in the architect's toolbox (and how about even finder or coarser granularity levels)? Are novel container patterns and technologies needed, or are established component and container models such as Spring Boot and Spring Cloud sufficient (see C. Richardson's "Microservices Chassis" pattern [25])? This set of research topics relates to PQs 6 and 7.

[6] This topic was for instance discussed in an ICWE 2016 WS-REST (un-)panel; the session notes are available at https://github.com/apiacademy/WSREST2016/wiki/Olaf-Zimmermann

[7] Assembly is a deliberately neutral term; related terms that were established earlier include service composition, business process management, and even workflow management.

Microservice integration and discovery. When accessing microservices conceptual dissonances and format/protocol mismatches must be overcome. While some of this work can be left to the Web machinery, leveraging dynamic content negotiation and supporting multiple media types in service requests and responses, it is not yet clear what the pendant to enterprise application integration in the microservices age is. The Enterprise Service Bus (ESB) pattern [33], its commercial implementations and their project use have been criticized by members of the microservices community as overly heavyweight, inflexible and unmanageable; however, the requirement for such integration capabilities cannot simply be argued away. Hence, message routing and transformation patterns [10] have to be supported and possibly adapted to fit the microservices tenets: Do emerging microservices patterns such as C. Robinson's "API Gateway" [25] provide sufficient design guidance or are additional ones needed? If so, how to stitch such patterns together? Should transformations be wrapped in and deployed as first-class microservices (of a particular type)? And once services (of various types, e.g., integration services vs. domain logic services) have been deployed, how can and should they be found, e.g., via network- or application-level discovery? These two research topics stem from, and can be traced back to, PQ 8.

Dependency management. Binary and source dependency resolution (static and dynamic) is needed and difficult to design, irrespective of build and integration technologies used.[8] Just one example: the transitive closure of open source licenses used in small projects, see e.g. the domain-driven-design sample application (realizing two cargo use cases) from 2009, can easily reference hundreds of libraries, which reference dozens of different license types directly or indirectly [3]. Hence, should a concept such as service wiring from Service Component Architecture (SCA) [24] be revived and possibly extended to support license- and QoS-aware microservice dependency management (in the context of the microservices tenet "fine grained service interfaces" and principle "independently deployable" from Section 3)? This problem has its roots in PQs 6, 7, and 8; its solutions can be seen as prerequisites for successful development and usage of service contract and deployment tools (and traced back to a tenet from Section 3, decentralized continuous delivery).

Service and end user/client application testing. Microservices usage promises to be more dynamic and flexible, requiring more runtime and configuration effort than coding. As a consequence, an application's external boundary gets blurred; therefore it is no longer clear where/how pre- and post-conditions can be specified and (validated during testing). How does dynamic, ad hoc service (provider) mocking work? Are integrated white box and black box service-specific test frameworks needed (a.k.a. "SUnit")? How to source realistic service invocation test data (in absence of end user oriented client applications)? What is the impact of continuous delivery, cloud computing, and DevOps on tests? Testing in production is an option in some, but not in all business sectors and application genres (example: video-on-demand provider vs. financial institution [2,7]). This topic originates from PQs 5, 7, and 8.

[8] Progress has been made in recent years; functionally rich (but sometimes cumbersome to use) various proprietary and open source package managers and integration servers are now available for programming languages and platforms such as Java, Ruby, Scala, and Linux.

5. Summary and Conclusions

In this paper, we distilled seven microservices tenets from the literature: fine-grained interfaces (to independently deployable services), business-driven development, IDEAL cloud application architectures, polyglot programming and persistence, lightweight container deployment, decentralized continuous delivery, and DevOps with holistic service monitoring.

A viewpoint-based analysis and comparison of two popular definitions of microservices followed, which supports the position that microservices are not entirely new, but qualify as "SOA done right". More precisely, microservices comprise an organic implementation approach to SOA (just like Scrum is one, but not the only way to practice agile development). Common characteristics include business orientation, polyglot programming in multiple paradigms and languages, and design for failure; decentralization and automation are emphasized specifically in the microservices implementation approach. An important microservices property is that services can be deployed independently of each other, which requires services to communicate with each other via remoting protocols such as HTTP and asynchronous message queues. The comparison tables in Section 3 as well as Table 4 provide evidence for this evolutionary and complementary positioning of microservices w.r.t. SOA.

Table 4. Summary of relationships: SOA style and its microservices implementation approach.

Topic (Concern)	**SOA Style**	**Microservices Implementations**
Core metaphor	Service, service consumer-provider contract pattern	Fine-grained service interfaces, independently deployable services, RESTful resources
Method	Object-Oriented Analysis and Design (OOAD); service-specific design methods	Domain-Driven Design (DDD), agile practices (refining and partially simplifying OOAD
Architectural principles	Layering, loose coupling, flow independence, modularity	IDEAL cloud architectural principles (overlapping with SOA principles, but also covering cloud computing-specific aspects)
Data storage	Information services, service provider implementations (e.g., RDB, backend system)	Polyglot persistence (SQL, NoSQL storage types, NewSQL)
Deployment and hosting	out of scope (of logical style definition)	Lightweight containers (e.g., Docker, Dropwizard); xaaS cloud offerings
Build tool chain	out of scope (of logical style definition)	Decentralized continuous delivery
Operations (systems management)		Lean but comprehensive system/service management (a.k.a. DevOps)
Message routing, transformation, adaption	Enterprise Service Bus (ESB) pattern	API gateways, lightweight messaging systems (e.g., RabbitMQ); transformation services
Assembly/composition	Service choreography and orchestration patterns	Service orchestration via Plain Old Programming (POP)
Lookup (runtime, design time)	Service registry pattern (including service repository)	Custom service registries and repositories (e.g. Swagger-based), service discovery (on application level and network level)

Technologies and software engineering practices have evolved since 2009 when the SOA hype had passed its peak and came to a temporary halt (e.g., cloud computing, NoSQL, and DevOps have become popular since then).

The complexities and fallacies of distributed computing cannot be argued, tested, or generated away, no matter how trends are named and positioned and no matter how much progress is made with computing, storage, and networking hardware, virtualization, containers, deployment automation; in the foreseeable future, requirements and constraints regarding accuracy, latency, scalability, security (of algorithms and data structures/logic and data access), etc. will continue to drive architectural decision making and implementation work on projects (and products). Hence, successful microservices realizations have to combine SOA principles and patterns with modern software engineering practices. The practitioner questions identified in Section 4 of this paper have to be answered to increase the chances that the microservices trend will sustain; related research opportunities abound.

In summary, service-orientation is here to stay, reconfirming the (frequently misinterpreted) blog post entitled "SOA is dead" [20], whose subtitle was: "long live services" – of various kinds and on multiple levels of granularity □

Acknowledgments

I would like to thank Jonas Biedermann, Simon Brown, Wolfgang Giersche, Gregor Hohpe, Hansjörg Huser, Oliver Kopp, Frank Leymann, Daniel Lübke, Cesare Pautasso, Gerald Reif, Mirko Stocker, the attendees of the SEI SATURN 2015 workshop on microservices as well as the anonymous peer reviewers of and paper presentation/poster session attendees at SummerSoC 2016 for inspiring discussions and/or thoughtful comments on earlier versions of this article. I would also like to thank James Lewis for the constructive and inspiring discussion following his ICWE 2016 keynote, which confirmed the SOA vs. microservices positioning presented in this paper, which had been accepted for publication previously.

References

1. Allamaraju, S., Describing RESTful Applications, http://www.infoq.com/articles/subbu-allamaraju-rest
2. Cockroft, A., Migrating to Microservices, http://www.infoq.com/presentations/migration-cloud-native
3. Evans, E., Domain-Driven Design – Tackling Complexity in the Heart of Software, Addison Wesley, 2003.
4. Fairbanks G., Keeling M., Microservices Workshop at SEI SATURN 2015, https://github.com/michaelkeeling/SATURN2015-Microservices-Workshop
5. Fowler, M., Microservices Prerequisites, http://martinfowler.com/bliki/MicroservicePrerequisites.html
6. Fowler M., Patterns of Enterprise Application Architecture. Addison Wesley, 2003.
7. Giamas, A., From Monolith to Microservices, Zalando's Journey, http://www.infoq.com/news/2016/02/Monolith-Microservices-Zalando
8. Haberle T., Charissis L., Fehling, C., Nahm, J., Leymann, F., The Connected Car in the Cloud: A Platform for Prototyping Telematics Services, IEEE Software 32(6), 2015
9. Hagen, K., Murer, S., Fifteen Years of Service-Oriented Architecture at Credit Suisse, IEEE Software 31(6), 2014
10. Hohpe, G., Woolf, B., Enterprise Integration Patterns. Addison Wesley, 2004.

11. Hüttermann, M., DevOps for Developers, Apress, 2012.
12. Jones, S., Microservices is SOA, for those who know what SOA is, http://service-architecture.blogspot.ch/2014/03/microservices-is-soa-for-those-who-know.html
13. Josuttis, N., SOA in Practice, O'Reilly, 2007.
14. Julisch, K., Suter, C., Woitalla T., Zimmermann, O., Compliance by Design – Bridging the Chasm between Auditors and IT Architects. In: Computers & Security, Volume 30, Issue 6-7 2011, Elsevier
15. Kruchten, P., Agile Architecture, blog post (with links to additional information), http://philippe.kruchten.com/2013/12/11/agile-architecture/
16. Kruchten, P., The 4+1 View Model of Architecture. IEEE Software 12(6), 1995
17. Lewis, J., Fowler, M., Microservices – a definition of this new architectural term http://martinfowler.com/articles/microservices.html
18. Little, M., SOA versus Microservices? http://www.infoq.com/news/2015/02/special-microservices-mark-litle
19. Loftis, H., Why Microservices Matter, https://blog.heroku.com/archives/2015/1/20/why_microservices_matter
20. Manes, A. T., SOA is Dead; Long Live Services, Burton Group, 2009. http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html
21. Mean.js, Open-Source Full-Stack Solution For MEAN Applications, http://meanjs.org/
22. Nilsson, J., Chunk Cloud Computing, http://jimmynilsson.com/blog/posts/CCC.pdf
23. Newman, S., Building Microservices – Designing Fine-Grained Systems, O'Reilly, 2015
24. OASIS, Service Component Architecture, http://www.oasis-opencsa.org/sca
25. Richardson, C., Microservices: Decomposing Applications for Deployability and Scalability, https://www.infoq.com/articles/microservices-intro
26. Rotem-Gal-Oz, A., Services, Microservices, Nanoservices – oh my! http://arnon.me/2014/03/services-microservices-nanoservices/
27. Spolsky J., Architecture Astronauts Take Over,, http://www.joelonsoftware.com/items/2008/05/01.html
28. Steinacker, G., Otto case study, https://dev.otto.de/author/gsteinacker/
29. Strumpflohner, J., Notes and thoughts on Martin Fowler's talk about Microservices at XConf, http://juristr.com/blog/2015/01/notes-microservices-fowler-xconf/
30. Torres. F., Context is King: What's Your Software's Operating Range, IEEE Software 32 (6), 2015
31. Wähner, K., Do Good Microservices Architectures Spell the Death of the Enterprise Service Bus?, https://www.voxxed.com/blog/2015/01/good-microservices-architectures-death-enterprise-service-bus-part-one/
32. Wiggins, A., The Twelve-Factor App, http://12factor.net/
33. Zimmermann, O., An Architectural Decision Modeling Framework for Service-oriented Architecture Design, University of Stuttgart, 2009.
34. Zimmermann, O., Wegmann L., Koziolek H., Goldschmidt, T., Architectural Decision Guidance across Projects, Proc. of. IEEE/IFIP WICSA 2015.
35. Zimmermann, O., Milinski, S., Craes, M., Oellermann F., Second Generation Web Services-Oriented Architecture in Production in the Finance Industry, Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '04). ACM, 2004.