This is a preprint of a paper accepted for publication at Euromicro SEAA 2018 in Prague.

A Collection of Software Engineering Challenges for Big Data System Development

Oliver Hummel*, Holger Eichelberger†, Andreas Giloj‡, Dominik Werle§ and Klaus Schmid†
*Faculty of Computer Science, Mannheim University of Applied Sciences, o.hummel@hs-mannheim.de
†Software Systems Engineering, University of Hildesheim, {eichelberger,schmid}@sse.uni-hildesheim.de
‡Department of Architecture-Centric Engineering, Fraunhofer IESE, Kaiserslautern, andreas.giloj@iese.fraunhofer.de
§Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology, dominik.werle@kit.edu

***Abstract*—In recent years, the development of systems for processing and analyzing large amounts of data (so-called Big Data) has become an important sub-discipline of software engineering. However, to date there exits no comprehensive summary of the specific idiosyncrasies and challenges that the development of Big Data systems imposes on software engineers. With this paper, we aim to provide a first step towards filling this gap based on our collective experience from industry and academic projects as well as from consulting and initial literature reviews. The main contribution of our work is a concise summary of 26 challenges in engineering Big Data systems, collected and consolidated by means of a systematic identification process. The aim is to make practitioners more aware of common challenges and to offer researchers a solid baseline for identifying novel software engineering research directions.**

I. Introduction

Big Data systems have become an important factor for information technology today. Thus, the development of such systems has become an important field with a significant impact on software engineering. In current literature, the focus is often on technical aspects [1] or specific platforms, whereas we take a different perspective, aiming at the process and methodological aspects, which have been rarely discussed to date. Knowing and understanding related issues, however, is important for arriving at a systematic approach and, in turn, rational decisions in developing Big Data systems [2]. Such systems are usually operating at the limits of data processing and technology and are therefore distributed systems. This class of systems is well-known to suffer from a variety of concurrency issues that significantly complicate their development [3]. Moreover, common approaches for building Big Data systems, such as the Lambda Architecture [4], call for at least a duplication of processing paths and hence increase system complexity even further from a software engineering perspective.

To the best of our knowledge, however, for the time being there exists no comprehensive survey or otherwise integrated discussion of specific software engineering challenges that arise from the development of Big Data systems. Hence, we start by identifying and discussing these challenges in order to make practitioners aware of them and give researchers some solid directions for potential future work in this area. The discussion we provide in this paper is guided by the collective background of the authors in engineering such systems both in industry and academic contexts (e.g., development of an industrial Big-Data-capable data cross-linking system*, a cross-organizational Big Data analytics platform in PRO-OPT [5]‡, a configurable and adaptive data processing infrastructure in QualiMaster [6]†), as well as based on consulting experience and literature. In line with our experience in engineering such systems, we focus on information systems for Big Data and exclude technically oriented Big Data systems such as factory automation systems. Nevertheless, we are confident that many of the lessons learned here will also apply in these other contexts.

The remainder of this paper is structured as follows: After explaining the process we applied to collect and analyze the challenges in Section II, we present and explain these in the main part of our paper, Section III. In Section IV, we discuss the most closely related works on this topic that we were able to discover. Finally, we round off our paper with a conclusion and a brief outlook on future work in Section V.

II. Research Process

As depicted in Figure 1, we identified and collected challenges using a systematic and collaborative process that roughly required one year.

We started off with a *discussion* of challenges, problems, and personal experience during a meeting of the German Computing Society (GI) Working Group on Big Data Architectures [7]. This was followed by an *initial collection* of challenges where the participating authors (the currently active core of the Working Group) filled a document template [1]. Various presentations and other input from non-permanent members of the Working Group complemented this initial data collection, which contained a description and a justification for each

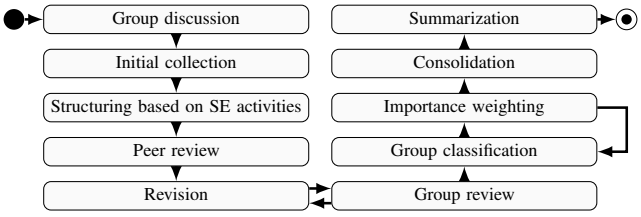

Fig. 1. Collaborative challenge identification and classification process.

[1]We used one MS Word document for each challenge with entries for "description", "experience or support why this is a challenge", and "references."

challenge, its relationship to Big Data architectures, as well as personal experience and references to existing literature.

As this collection was driven by experience, we then aimed at structuring and classifying the initial results according to common activities of the *software development life-cycle* and structured Section III along these phases. This happened during a face-to-face meeting of the Working Group. The structuring revealed additional challenge candidates, which were then assigned to group members for detailing according to a revised document template. All documents were *peer reviewed* by another group member not involved in the initial description of the respective challenge.

After the original authors had revised the reviewed challenges, the entire *group reviewed and discussed* the outcomes during another face-to-face meeting. This led to an aligned summary of potential challenges for phase. Afterwards, these challenges were subjected to another *review round* including discussion and revision. As a result, we identified several candidates as duplicates or as not specific for Big Data systems. The collection process ended with a revision of the candidate *classification* as well as the classification structure.

To summarize the results, all authors individually evaluated the *importance* of the candidates and discussed the results during several online meetings. This was done in a spread sheet with rows of potential challenges and one column per author. Candidates with diverging ratings were subject to consensus *discussions*, which led either to agreement or, partly, to a re-classification or clustering of similar challenges. Finally, we *consolidated* and *summarized* the results for this paper.

III. Challenges

In this section, we discuss the classification of the challenges we obtained from the process explained in Section II. Table I provides an overview of the individual challenges and how we allocated them to the common software development life-cycle. In the remainder of this section, we will discuss each identified challenge as listed in the table and summarize the challenges, focusing on crosscutting concerns, in Section III-E.

A. Project & Requirements Management

Managing Big Data projects as well as eliciting and managing requirements for Big Data projects is even more challenging than usual, as novel analysis opportunities may arise from the available data. Additionally, various interdisciplinary competencies must interact to process and analyze the data and deal with potential problems of statistical soundness or data privacy that may impact the results. In the following, we will discuss the respective challenges we identified.

Unclear Requirements (PM1). In the development of Big Data systems, desired outcomes are often unknown at the beginning as stakeholders can neither imagine the capabilities and the potential of analyses nor their future desires inspired by using the system under development [8]. This imposes challenges on the involved parties as well as the applied methods: To some degree, this turns requirements engineering upside down, as data analysts have to explain as early as possible which analyses are possible and feasible at all or are better avoided, e.g., due to issues with statistical soundness (cf. PM7) or data privacy (cf. PM5). Furthermore, as described below, new requirements and ideas might continue to appear during realization, testing, and operation of a system, making a highly agile or even exploratory management and development approach an absolute necessity.

Project & Requirements Management	
PM1	Unclear Requirements
PM2	Emergence of New Requirements from Data
PM3	Highly Interdisciplinary Teams
PM4	Integration of Hard- and Software Components
PM5	Privacy Implications
PM6	Complex Trade-offs between Quality and Performance
PM7	Prevention of Self-Fulfilling Prophecies
Architecture & Development	
AR1	Distribution and Concurrency
AR2	Steep Learning Curves due to Novelty of the Field
AR3	Query-Driven and Evolutionary Design
AR4	Lack of Unified Modelling Support
AR5	Algorithmic Idiosyncrasies
AR6	Duplicated Implementations
AR7	Data Consistency and Availability
AR8	System vs. Platform Development
AR9	Runtime Adaptability and Platform Independence
Quality Assurance	
QA1	Challenging Visualization and Explainability of Results
QA2	Non-Intuitive Notion of Consistency
QA3	Complex Data Processing and Different Notions of Correctness
QA4	High Hardware Requirements for Testing
QA5	Difficult Generation of Adequate, High-Quality Data
QA6	Lack of Debugging, Logging, and Error-Tracing Methods
QA7	State Explosion in Verification
QA8	Ensuring Data Quality
Deployment & Operations	
DO1	Complex "Elastic" Provisioning
DO2	Complex Monitoring

TABLE I
Summary of Identified Challenges.

Emergence of New Requirements from Data (PM2). Observations from data can give rise to new requirements or refine existing requirements. This can pave the way for new functionality or a different realization of some functionality [8]. Sometimes this can only be realized during operation of a system, leading to late requirements changes. Hence, it may be basically not possible to define all requirements before the actual system is running. Even more challenging, from a requirements management point of view, is the fact that *data can become a new stakeholder*, as new data might facilitate new analysis opportunities [9]. However, existing Requirements Engineering (RE) methods that mostly focus on early object-oriented analysis of a finite data set with a known structure are not prepared to deal with this situation.

Highly Interdisciplinary Teams (PM3). Creating novel and successful data analysis systems requires the formation and management of highly interdisciplinary teams. This involves integrating technical and data analysis competencies as well as competencies in, e.g., sociology, ethnography, cognition, psychology, or jurisdiction [8]. Involvement of external consultants is helpful, in particular when setting up first projects, however, for long-term success, internal competence is indispensable. Moreover, the formulation of data analysis tasks is particularly hard for users who are not data analytics experts and who are

not familiar with common data analysis concepts and algorithms, so this requires more communication and management.

Integration of Hard- and Software Components (PM4). The setup of Big Data systems often requires integration of different frameworks, e.g., to implement the Lambda Reference Architecture [4], [10]. Also, Big Data processing often cannot be accomplished by software alone. Hardware co-processors [11], [12] such as Graphical Processing Units (GPUs) are widely used as a cost-effective, powerful, and scalable platform for various calculation-intensive tasks. Combining hard- and software-based processing into hybrid systems [11], [12], [13] is challenging, as it requires teams with broad technical knowledge ranging from hardware selection or even design via fast networking technologies to software and integration capabilities. This adds even more pressure for interdisciplinary collaboration as mentioned in PM3.

Privacy Implications (PM5). Data analyses using and integrating personal data must be compliant with numerous data protection guidelines, data licenses (if applicable) and (changing) laws, e.g., by using encryption, partitioning, anonymization, de-personalization or pseudonymization techniques, ans may also be subject to negotiations with works councils [8]. To some degree, a safe strategy here may be overly restraining, as it may not always be obvious to legal laymen like developers what forms of analyses may be allowed with the collected data. Data protection is also a rapidly developing field, i.e., it challenges lawyers or even requires specific proactive compliance officers who monitor legal changes and recognize necessary posterior changes to existing systems.

Complex Trade-offs between Quality and Performance (PM6). Carefully applying proven design tactics often allows scaling traditional information systems in a relatively hassle-free manner. For Big Data systems, however, the combination of huge data sets with large-scale distribution and the trade-off between consistency, availability, and partitioning known as CAP theorem (cf., e.g., [4]) usually leads to more difficulties in scaling them. Quite often, this even requires stakeholders to abandon their expectations of perfect quality as merely approximate, but faster algorithms or (batch) algorithms with higher latency can be used. Clearly, this makes it very hard to predict how a system will behave in terms of throughput or latency. This, in turn, makes sizing and planning the deployment of Big Data systems more difficult [14] or even unpredictable and hence may imply additional prototyping and testing, e.g., with very high or changing loads (cf. QA4, QA5).

Moreover, in Big Data systems the implementation paradigm has a significant impact on performance, latency, or quality of results. For instance, a batch algorithm constrains the degrees of freedom in the implementation, comes with a high degree of latency, and is hard to use with current approaches for performance analysis such as Palladio [15]. The latter often do not support Big Data frameworks and their properties as modeling concepts, in particular not for estimating the effects of exchanging similar frameworks or scaling them [14]. In the context of cloud based data-intensive systems, these approaches (like [16]) are still confronted with typical problems, such as complex model creation for middlewares or resource demands of rare events.

Prevention of Self-Fulfilling Prophecies (PM7). In order to prevent misinterpretations of results [17], [18] or even self-fulfilling prophecies in creating them, the design of data analysis approaches requires the application of sound scientific methods. This demands discipline from all involved parties to prevent data dredging [18] and to properly apply statistical methods, e.g., by identifying hypotheses, collecting data, experimenting, and identifying evidence [8].

B. Architecture and Development

Big Data systems raise a number of practical challenges for architects and system designers alike. The majority of these challenges is caused by the fact that Big Data systems are distributed and that the field itself is rather new so that its tools and frameworks are often evolving rapidly. However, there are additional challenges specific to their design and development, as we will elaborate below.

Distribution and Concurrency (AR1). Big Data systems are pervasively distributed / concurrent as they belong to the class of systems that cannot process the accruing amount of data on a single machine [19]. As a consequence, they inherit all problems already known from the development of such systems [3], like the difficulties in establishing a consistent understanding of the system state or dealing with partial system failures. Hence, we will not reiterate these well-known challenges any further, unless we identified specific aspects related to Big Data.

Steep Learning Curves due to Novelty of the Field (AR2). While traditional information systems typically use a proven stack of technologies, Big Data systems are very much driven by the job at hand and hence often force developers to come up with a novel combination of frameworks and technologies. Each of these might already have a steep learning curve, but bringing them together and orchestrating their interplay is a novel challenge every time. Although reference architectures, such as the Lambda Architecture [4], [3], and their understanding have started to mature, more work is necessary as implementation technologies still suffer from a lack of standardization, large evolutionary steps of frameworks, or even the rise of completely new technologies. Consequently, there exists a lack of in-depth understanding of Big Data architecture tactics and patterns as well as for implementation stacks and, in turn, for deploying them to scalable environments, e.g., a cloud. This causes constant pressure for architects to experiment with and learn new implementation approaches.

Although the community has been aiming at collecting Big Data patterns for some time [20], this research currently remains in a rather immature state and is often based on a few anecdotal reports (such as blog posts) since there is simply not (yet) enough publicly available information on how to successfully build Big Data systems in order to derive well-established patterns for them. On this basis, it is clearly hard to predict how an architecture should be defined in order to

address certain goals regarding performance / scalability / etc. and how to integrate various different processing paradigms.

Query-Driven and Evolutionary Design (AR3). In traditional information systems, where "merely" the typical create-read-update-delete (CRUD) operations must be implemented, approaches such as O/R mapping have matured and designing the persistence layer of a system has become more routine than particularly challenging. In Big Data systems, however, data access and subsequent data processing often generate the core of the expected value. As these systems usually come close to technological limits, data handling has a much more profound impact on the architecture and a query-driven approach is needed [9]. Even worse: as the desired value is often not clear in advance (PM1, PM2), the core of architectural design is often driven by queries discovered after requirements analysis, by prototyping large parts of the architecture. Thus, major architectural decisions are often not possible in time or need to be made in a highly evolutionary manner. Current software engineering methods widely lack support for the creation of such "query-driven agile architectures".

Lack of Unified Modelling Support (AR4). Common modeling languages such as the UML or the E/R diagram notation provide little off-the-shelf support for the use of Big Data concepts. Although the UML can be extended via custom metamodels, there still exists a lack of standardized extensions, let alone experience with their application. Simultaneously, although there exist various modeling notations for Big Data tools and frameworks (e.g., pipe diagrams for MapReduce [4], pipelines for streaming [21], or the DICE UML profiles [16]), they lack standardization and support from common CASE tools. Moreover, Big Data applications are often developed in a very interdisciplinary fashion (see PM3) so that creating commonly accepted modeling approaches and avoiding a Babylonian confusion with notations from areas such as Data Science is clearly another challenge.

Algorithmic Idiosyncrasies (AR5). Big Data frameworks often require the application of programming paradigms unfamiliar to many developers, e.g., MapReduce on immutable data [4]. These (novel) paradigms necessitate a different way of solution design thinking and often impose limitations on architectural degrees of freedom. Moreover, Big Data algorithms must be horizontally scalable (cf. DO1), which generally prohibits or seriously limits the use of algorithms with more than linear time or space complexity. For example, limited processing time (PM6) may imply the use of special versions of algorithms (e.g., HyperLogLog), which may lead to a loss in result quality. Here, the challenge is to balance understandability, result quality, and processing time for the required design compromises.

Duplicated Implementations (AR6). As it is not always possible to sacrifice quality for performance, the most popular Big Data reference architecture, the Lambda Architecture [4], proposes supporting both in a time-displaced manner. This comes at the price of implementing functionality twice: once for the so-called (streaming) speed layer, where time is more important than quality, and once for the so-called batch layer, where quality is more important than time. Quite frequently, the algorithms are different (cf. AR5) and both layers even need their own specialized persistence so that saving data is also implemented twice. Sometimes, when special and complex analyses must be executed, it can become even more challenging to meet all performance requirements. Here, the use of a so-called polyglot persistence [22] has become a common practice. However, it comes at the price of data and code duplication or even multiplication. Another aspect regarding sufficiently fast data processing may be the need for priority lanes, where important data must be "smuggled" into the processing unit quickly, which again adds complexity to the design. So the challenge is to realize different versions of the same algorithms consistently, e.g., based on appropriate model-based approaches (AR4).

Data Consistency and Availability (AR7). As if implementing polyglot persistence alone would not be challenging enough, it comes with consistency problems between different storage systems; increasing the consistency issues within a single storage system in usual distributed systems. The intense discussions around the CAP theorem and the use of "soft state" in NoSQL databases underline [23] the importance of this issue. However, these discussions widely revolve around solving consistency and availability within one persistence under heavy load [19], not within a zoo of numerous systems with different characteristics. Consequently, dealing with state and consistency in polyglot persistence is a challenge that needs to be tackled urgently.

System vs. Platform Development (AR8). While we typically speak of Big Data *systems*, their development often rather resembles that of a platform in a software ecosystem [24]. This can have two reasons: (a) Planned development as a platform: the significant investment for Big Data systems can often only be justified if a number of applications can benefit from it. The Big Data system is then designed as a platform from the beginning; (b) development as a platform due to uncertainty: Here a specific system is envisioned, but due to significant uncertainty (PM1, PM2), the development is separated into a generic platform and specific parts on top. Sometimes, there is also a hybrid way: Platforms like Twitter and Facebook were initially conceived as closed systems, but transformed into platforms over time. Thus, successful Big Data systems may need to address challenges known from variability management (e.g., [21]) or the design of long-lasting systems.

Runtime Adaptability and Platform Independence (AR9). Platform-independent development has long been a challenge in software engineering, and there is still no satisfactory solution some 15 years after Model-Driven Architecture (MDA) was proposed. The solution has not become any better for Cloud environments in which Big Data applications are frequently deployed. On the contrary: vendor-specific tools and hence the fear of vendor lock-in are a common problem in this context. Thus, the DICE project [16], e.g., envisages Big Data application models that are technology-agnostic.

Cloud computing marketing and the need to cope with varying data loads have led to another desire, namely the ability to make applications seamlessly scalable or "elastic". From an

architecture point of view, this requires additional monitoring capabilities, an adaptation engine capable of deriving insights from monitoring, and means for automatically adapting the system, e.g., by provisioning new hardware at run-time.

C. Quality Assurance

In the previous section, we focused on the construction of Big Data systems. Now we target the follow-up question: How to ensure the quality of such a system? As quality assurance is a crosscutting challenge, we have condensed aspects regarding testing, usability, and data quality in the following.

Challenging Visualization and Explainability of Results (QA1). As discussed in the literature [25], visualization of large amounts of data, in particular with high dimensionality (as it comes from complex or heterogeneous data sets), is notoriously hard [9]. Here, finding the right combination of dimension and resolution for result visualizations is important, as it enables the user to derive insights and, in turn, to assess the validity of the results. Real-time interaction with visualizations—such as changing the time interval of the displayed data or changing the displayed dimensions—can help users understand the data, but can also provide feedback on performance issues (cf. PM6) or increase system complexity (cf. AR6). Even if the analysis results are presented in an understandable way, the detailed process of how an analysis result was produced might neither be easy to grasp nor easy to trace even for experts. This becomes worse if decisions are made, for instance, by artificial intelligence approaches such as deep learning. Hence, trustworthiness and understandability of data, processing, and analysis results is a particular challenge [26] that is strongly interrelated with engineering the system.

Non-Intuitive Notion of Consistency (QA2). Because Big Data systems may trade consistency for availability in a network partition situation (cf. AR7) or for performance in general (cf. PM6), "eventual consistency" may be a feasible resort. However, (temporary) inconsistency can be confusing to users and quality engineers alike, for example when updates made to the data are not visible in subsequent requests. Here the challenge is to handle (in)consistency in an understandable manner, e.g., by ensuring that a user at least sees his own recent changes [3]; however, this may lead to performance issues (cf. PM6) or increased system complexity (cf. AR6).

Complex Data Processing and Different Notions of Correctness (QA3). Big Data processing is typically complex, e.g., due to a lot of interdependencies among individual processing steps. Hence, it is very difficult to determine whether an operation is (in)correct, as the influence of a single operation on the overall result may be comparatively small. Due to the complexity of the data processing and the computations itself, it may even be difficult to determine adequate test results. This leads to a paradox where it is not possible to have precise data to test against. Consequently, testing for plausibility becomes a typical fallback. The challenge is to design simple control mechanisms that function with a vague indication of correctness while still recognizing incorrect implementations that are testable with today's testing capabilities.

High Hardware Requirements for Testing (QA4). Considering the main characteristics of Big Data systems (volume, velocity, and variety), distributed processing and high workloads are imperative. In practice, this results in systems often failing due to unexpected small issues such as a lack of storage space. Thus, thorough testing of Big Data systems requires a similar workload as will be used in the real system, and testing also issues such as parallelization, performance, scalability, etc. For this purpose, an appropriate test system comparable to the production system should be available [27]. However, in practice this is often not possible as costs may be prohibitive or a large enough cluster of machines is simply only available for production. As a consequence, some part of the testing may only be possible on the production hardware, which can delay development. Even worse, testing a real-life workload (see also QA5 below) may have to happen during actual operation, hence requiring special precautions and risk management to avoid impacting the production system. Moreover, testing a dynamically scalable, but potentially immature system is a severe financial risk when relying on on-demand cloud services. Thus, testing Big Data systems in realistic settings is an actual issue for product quality and quality assurance.

Difficult Generation of Adequate, High-Quality Data (QA5). Testing Big Data systems requires realistic high-quality data sets [28]. While large volumes of data may be created by multiplying smaller data sets, terabytes of storage must be provided for less regular data sets. Moreover, relying merely on data for volume-related tests may be too restrictive to cover all relevant phenomena of interest; i.e., data for the other "Big Data Vs" such as variety and veracity (if necessary, including different kinds of data such as documents or videos) must also be provided. In summary, creating and handling realistic application-specific test data sets covering all relevant characteristics is a practical and methodological obstacle.

Lack of Debugging, Logging, and Error-Tracing Methods (QA6). Due to their distributed nature, Big Data systems must ultimately be tested in a distributed environment (cf. AR1, QA3). However, the currently limited capabilities for distributed development tools and debugging [9] impose another difficulty—developers using Big Data frameworks often have to rely on distributed log files. This complicates processing, and understanding the information may require merging the log files and ultimately calls for advanced distributed debugging and log visualization approaches, e.g., [29].

State Explosion in Verification (QA7). The use of growing clusters for Big Data processing particularly increases the complexity in applying verification approaches due to a combinatorical state explosion. Although symbolic model checking or partial reduction techniques have led to a breakthrough in practical verification, explicit state models are still used due to their better verification capabilities [30]. So applicable verification approaches for distributed computing and, in addition, for hybrid data processing (cf. PM4) are needed.

Ensuring Data Quality (QA8). For Big Data systems, the metaphor "Data are the crude oil of the future" is often used. Nevertheless, data on its own is only of limited significance;

it becomes meaningful only when its quality, e.g. in terms of completeness or consistency, is proven. However, assessing data quality in the context of Big Data is neither semantically nor computationally trivial [31], in particular if data (of different types) is aggregated or merged during processing.

D. Deployment & Operations

Big Data systems imply distributed data storage and processing and often also come with requirements for long-running and continuous operation. This makes both the provisioning and the monitoring of the composed system challenging.

Complex "Elastic" Provisioning (DO1). As Big Data systems must be scalable on demand, they are often deployed to a cloud environment. However, privacy, legal, and licensing issues for commercial components or data (cf. PM5) may limit the possible environments; in QualiMaster, e.g., data licenses required processing on local infrastructures [32]. Flexible "elastic" deployment mechanisms are a key part [19] to cope with the complexity of distribution, component configurations, and components that are similar but not fully substitutable, as in polyglot persistence (cf. AR6). Additionally, virtual machine and container technologies increase the complexity, as they require tight integration of development, deployment, and operation (DevOps). Moreover, long-running systems may require migration of the system between alternative cloud platforms over time, which is often limited or even prevented by provider specific environments. Besides the impact of moving data during migration, vendor lock-in situations may be prevented through standardization efforts by all involved parties or model-driven approaches that allow generating / migrating the required functionality based on an abstract specification (cf. AR4). In summary, provisioning Big Data systems is a challenge covering technical, legal, and standardization issues.

Complex Monitoring (DO2). Monitoring of the operations of Big Data systems needs to cover the involved resources, the processing, and the processed data [33]. To some degree, Big Data frameworks already include monitoring mechanisms and provide dashboards (e.g., Apache Storm) that enable the data engineer to oversee and optimize operations. Typically, existing solutions are not comprehensive; e.g., in [34], [35], further mechanisms were integrated to provide an overview of the relevant system state, including the underlying hardware and operating system. Moreover, the metrics used (cf. QA8) are usually neither standardized [34], [36] nor comparable across different frameworks in a cluster. Thus, encompassing, flexible, and low overhead monitoring of Big Data systems is an open issue, but also a hurdle for advanced processing approaches involving, e.g., self-adaptation (cf. AR9).

E. Summary

In this section, we will summarize the insights gained from the previous sections with a special focus on cross-cutting concerns that may have an impact on multiple stages or activities of the software development process.

Explorative Development Style (S1). Various questions that directly influence the success of a project are likely not to be answered during the planning of the system due to unclear requirements or immature tools and frameworks (PM1, PM2, AR3). Thus, the development process is highly exploratory and new requirements as well as design or even architectural decisions may occur at any time. Consequently, urgent changes may frequently impact all development activities, meaning that appropriate methods are necessary to deal with potentially high-impact changes. While exploratory processes have been a research objective in the agile and requirements community for some time, there still exists the need for further work in this direction, especially when it comes to software architecture.

Big Data Idiosyncrasies (S2). Avoiding the need for locking data under heavy write loads is one of the central hallmarks of a scalable architecture, which often comes at the price of sacrificing hard consistency constraints among multiple nodes or multiple persistences (cf. AR7, QA2). For example, in eventual consistency, replicated or redundant data is allowed to be inconsistent during certain time intervals; however, the data is guaranteed to reach a consistent state at some point. Another dimension of consistency exists among different computation paths in the system, e.g., between streaming and batch processing in a Lambda Architecture. Here, approximate results are calculated on incoming streaming data; these results are known to be overwritten by an exact result from a batch computation on the whole body of data at a later point. A natural extension of this concept is when multiple computation paths calculate related insights from different data based on polyglot persistence. This becomes particularly challenging when some of the paths may fail, e.g., due to software failures or incomplete data, and reintegration is necessary. While such weaker notions of consistency have become common in recent years [22], to our knowledge there is a lack of research on how to systematically plan and construct solutions that guarantee an eventually consistent state in complex system configurations.

Security (S3). Data security and privacy is a challenging aspect in the engineering of Big Data systems because the very purpose is to collect and analyze as much data of value as possible. Recent frameworks hence often sacrifice security for performance; i.e., they delegate security mechanisms to the environment so that a single successful attack can expose a large amount of data at once. Since a Big Data system is usually composed of a multitude of heterogeneous systems, such as database systems, or analysis middleware, developers need to consider all of the individual systems' attack vectors. The variety of the data requires additional care when importing and analyzing data from external sources, such as users, to prevent injection attacks, e.g., when interpreting documents that contain executable scripts. Privacy concerns (cf. PM5) are amplified because Big Data is usually comprised of a multitude of data sources for which it is unclear, whether a combination allows inferring personal information.

Hardware Availability (S4). For Big Data systems, the hardware is different than for traditional information systems since a larger amount of hardware is required to operate the final system. Nevertheless, various roles in the development process, such as algorithm designers, deployment specialists,

and testers, need to work with large clusters on a regular basis in order to evaluate and optimize their work. So managing the availability of hardware resources is certainly an issue that might influence project planning (cf. PM6, QA4).

Long-Term Operation (S5). The environments in which Big Data systems are running and the systems themselves are changing over time. These changes may be corrective, if the implemented software system needs to be upgraded, or adaptive, when the circumstances, e.g., available computing needs or capabilities, change or additional data sources are to be integrated. In particular, this includes the need for elasticity (cf. DO1). Such changes require tactics for non-disruptive operation of a system because it is not feasible to buffer the incoming data over extended periods of time and replay it into the system. Nowadays, such non-disruptive runtime operations are often not available. One example: although version 0.95 of Apache Storm allows changing the resources of a processing pipeline at runtime, this causes a complete restart of the data processing, implying a downtime of up to several minutes until the re-deployed algorithms are ready to run. Realizing an encompassing set of relevant runtime stabilization mechanisms for Big Data processing is an architectural challenge requiring the integration of further tactics, such as replication, which are not commonly provided by current frameworks. Applying these tactics, either as manual administrative operations or automatically in terms of a self adaptive system, requires observing a set of reliable metrics delivered by monitoring (DO2). Moreover, planning automated actions (AR9) to sustain the operations in Big Data processing, e.g., along the data flow in stream processing [32], [37], is subject to ongoing work.

The underlying configuration of the system can be an issue for long-term operations as well. Installing current Big Data systems typically includes configuring various frameworks, such as Apache Spark, Apache Cassandra, or Apache Kafka. These frameworks in isolation can already have a large number of configuration options and runtime parameters, which might have an unknown effect on the required resources and on thir performance. Moreover, there is often little or even no documentation on the installation of framework combinations, e.g., regarding which versions are known to work together or how frameworks affect each other in terms of stability or performance. In addition, configuration settings across frameworks must be done consistently, or services must be started in the right sequence to achieve stable cluster operations. In particular, this involves determining the required resources initially (cf. PM6) or over time. Some of the basic problems could be solved through a more standardized, e.g., architecture-driven, documentation, which would allow clashes between assumptions to become obvious at an early stage. Nowadays, optimizing the various configuration settings and files for different frameworks involved in a Big Data system is a manual task for an administrator. However, overseeing the configuration space and finding an optimal configuration requires systematic modification of the settings, benchmark runs, and experiment result analysis. Such an exploration of the configuration space could also be done automatically and optimized with the help of Machine Learning techniques. Nevertheless, a necessary prerequisite for this is formalized configuration knowledge, so that impossible and undesired combinations are not analyzed by such a configuration optimizer.

IV. Related Work

Although a significant body of work has been published on Big Data systems, it is mostly on technical solutions for particular problems and not on the specific challenges in Big Data Software Engineering. Due to space restrictions, we only provide a broad overview of the few publications naming such challenges and briefly link them to our work where similarities and other points of contact exist.

In [38], a summary of interviews with three industrial Big Data experts is presented, which include challenges as well as do's and don'ts from their personal experience. Related challenges target privacy (PM5), quality trade-offs (PM6), data consistency (AR7, S2), state management (AR7, S5), componentization (AR8), adaptivity (AR9, S5), scalability (S5) or user understanding (QA1). As the authors did not focus on any systematic identification and classification, they merely mention a sub-set of the challenges presented in our work. This is similar to the challenges, such as high and write-heavy workloads (AR7, S2) or required elasticity (DO1), listed by [19] or the NESSI consortium [39] (e.g., AR2, AR4, QA7, or DO2), which are intended as broad starting points as well. The authors of [40] highlight some challenges, found, e.g., in S3, PM3, or AR5 of our list, alongside the classic waterfall phases.

Chen et al. discuss [41] challenges such as interdisciplinarity (PM3), confidentiality (PM5), and scalability (S5), but also several data-related topics such as data acquisition (QA5). Jagadish et al.'s [1] list includes privacy and data ownership (PM5), timeliness (PM6), interdisciplinarity (PM3), and visualization of result data (QA1). Moreover, there is some overlap in our engineering challenges with [41] as well as [1], although we do not focus on specific data (processing) challenges as these works do. Madhavji et al. [27] list some example research challenges for software development, e.g. the need for more specific requirements methods (PM1) or the currently limited knowledge of Big Data reference architectures (PM4, AR2).

Moreover, there is related work with a rather specific focus. Among them, Heinrich et al. [14] discuss limitations and challenges for performance prediction for Big Data systems (subsumed by PM6). Fan et al. [42] discuss challenges of Big Data analysis from a statistical point of view, while Kaisler et al. [43] discuss some general issues related to software engineering, including unclear requirements (PM1), updating the system (S5), storage trade-offs, data completeness, quality and quantity trade-offs (PM6), data ownership, legal compliance (PM5), security (S3), scalability (S5), and distribution (AR1).

In summary, although there is quite a body of work mentioning "Big Data challenges", to the best of our knowledge, there exists no work yet to date that focuses on a comprehensive in-depth collection of development challenges for Big Data systems from a software engineering perspective.

V. Conclusion and Future Work

In this paper, we presented 26 challenges that are unique or at least significantly exacerbated in the context of developing Big Data systems. We collected and classified the challenges using a systematic collaborative process and categorized them along software development phases as follows: 7 Project & Requirements Management challenges, 9 Architecture & Development challenges, 8 Quality Assurance challenges, 2 Deployment & Operations challenges, and 5 cross-cutting challenges. This initial collection may not yet be complete; however, from our point of view, it provides the first broad overview of Big-Data-related challenges in software engineering. Thus, it might be used as a starting point for further research in this field. Although some of the issues listed in this paper are likely to be solved by the expected development of the field over time, a lot of research will be needed to solve the remaining immanent challenges. Moreover, there is a need for detailing and extending the challenges reported here in a more verbose form. Based on this, we aim to provide a long-term research agenda that shall help to address the immanent challenges systematically, e.g., through architectural patterns and tactics for the development of Big Data systems.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) *GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation*, by the European Commission (EC) project "QualiMaster" (grant 619525), and by the German Federal Ministry for Economic Affairs and Energy (BMWi) project "PRO-OPT", part of the technology program "Smart Data – Innovations in Data" (grant no. 01MD15004E). Any opinions expressed herein are solely those of the authors and do not reflect the opinion of the DFG, the EU, or the BMWi.

References

[1] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, Jul. 2014.

[2] M. Beyer, "Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data," *URL: http://www.gartner.com/newsroom/id/1731916*, 2011.

[3] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly, 2017.

[4] N. Marz and J. Warren, *Big Data – Principles and best practices of scalable data systems*. Manning Publications, 2014.

[5] PRO-OPT, "PRO-OPT - Big Data Production Optimization in Smart Ecosystems." [Online]. Available: http://www.pro-opt.org

[6] QualiMaster, "QualiMaster." [Online]. Available: http://qualimaster.eu/

[7] GI Arbeitskreis Big Data Architekturen, last visited: Feb. 2018. [Online]. Available: https://ak-bda.gi.de

[8] D. A. Marchand and J. Peppard, "Why IT Fumbles Analytics," *Harvard Business Review*, Jan. 2013.

[9] K. M. Anderson, "Embrace the challenges: Software engineering in a big data world," in *Intl. Works. on Big Data Software Engineering*, 2015.

[10] H. Cervantes and R. Kazman, *Designing Software Architectures*. Pearson, 2016.

[11] G. Lee, B.-G. Chun, and R. H. Katz, "Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud," in *HotCloud'11*, 2011.

[12] K. Shirahata, H. Sato, and S. Matsuoka, "Hybrid Map Task Scheduling for GPU-based Heterogeneous Clusters," in *Intl. Conference on Cloud Computing Technology and Science*, 2010.

[13] C. Qin and H. Eichelberger, "Impact-minimizing runtime switching of distributed stream processing algorithms," in *Intl. Workshop on Big Data Processing - Reloaded (BDPR'16)*, 2016.

[14] R. Heinrich, H. Eichelberger, and K. Schmid, "Performance Modeling in the Age of Big Data," in *Intl. Workshop on Interplay of Model-Driven and Component-Based Software Engineering*, 2016.

[15] R. H. Reussner, S. Becker, J. Happe, and al., *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016.

[16] G. Casale, D. Ardagna, M. Artac, F. Barbier, and al., "Dice: Quality-driven development of data-intensive cloud applications," in *IEEE/ACM 7th Intern. Workshop on Modeling in Software Engineering*, 2015.

[17] A. Reinhart, *Statistics Done Wrong: The Woefully Complete Guide*. No Starch Press, 2015.

[18] R. Nuzzo, "Statistical errors," *Nature*, vol. 506, 2014.

[19] I. Gorton, A. B. Bener, and A. Mockus, "Software engineering for big data systems," *IEEE Software*, vol. 33, no. 2, 2016.

[20] Arcitura™ Education Inc., "Big Data Patterns," last visited 27.1.2018. [Online]. Available: http://www.bigdatapatterns.org

[21] H. Eichelberger, C. Qin, and K. Schmid, "Experiences with the Model-based Generation of Big Data Applications," in *Big Data Mgt. Systems in Business and Industrial Applications (BigBia '17)*, 2017.

[22] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 2012.

[23] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, no. 3, 2008.

[24] S. Jansen, S. Brinkkemper, and M. A. Cusumano, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Edward Elgar Publishing, Incorporated, 2013.

[25] Wang, Lidong and Wang, Guanghui and Alexander, Cheryl Ann, "Big data and visualization: Methods, challenges and technology progress," *Digital Technologies*, vol. 1, no. 1, 2015.

[26] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, Feb 2015.

[27] N. H. Madhavji, A. Miranskyy, and K. Kontogiannis, "Big picture of big data software engineering: With example research challenges," in *Intl. Workshop on BIG Data Softw. Eng. (BIGDSE '15)*, 2015.

[28] A. Alexandrov, C. Brücke, and V. Markl, "Issues in Big Data Testing and Benchmarking," in *Intl. Workshop on Testing Database Systems*, 2013.

[29] I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst, "Debugging distributed systems," *Communications of the ACM*, vol. 59, no. 8, 2016.

[30] M. Camilli, "Formal verification problems in a big data world: towards a mighty synergy," in *ICSE Companion Proceedings*, 2014.

[31] M. Kläs, W. Putz, and T. Lutz, "Quality evaluation for big data: A scalable assessment approach and first evaluation results," in *Intl. Workshop on Software Measurement*, 2016.

[32] H. Eichelberger, C. Qin, K. Schmid, and C. Niederée, "Adaptive application performance management for big data stream processing," *Softwaretechnik-Trends*, vol. 35, no. 3, 2015.

[33] L. J. Bass, I. M. Weber, and L. Zhu, *DevOps - A Software Architect's Perspective*. Addison-Wesley, 2015.

[34] S. Agarwal, "Monitoring and Troubleshooting Apache Storm - DZone Big Data," 2016. [Online]. Available: http://bit.ly/2L4dYwX

[35] H. Eichelberger, C. Qin, and K. Schmid, "From resource monitoring to requirements-based adaptation: An integrated approach," in *Intl. Workshop on Monitoring Large-Scale Software Systems (MoLS'17)*, 2017.

[36] M. Alrifai, H. Eichelberger, C. Qin, R. Sizonenko, S. Burkhard, and G. Chrysos, "Quality-aware Processing Pipeline Modeling," 2015, QualiMaster Deliverable D4.1, http://qualimaster.eu.

[37] K. G. S. Madsen and Y. Zhou, "Dynamic Resource Management In a Massively Parallel Stream Processing Engine," in *Intl. Conf. on Information and Knowledge Management (CKIM'15)*, 2015.

[38] C. Szyperski, M. Petitclerc, and R. Barga, "Three experts on big data engineering," *IEEE Softw.*, vol. 33, no. 2, Mar. 2016.

[39] A. Metzger (ed.), "Software engineering: Key enabler for innovation," *NESSI White Paper*, 2014.

[40] C. E. Otero and A. Peter, "Research directions for engineering big data analytics software," *IEEE Intelligent Systems*, vol. 30, no. 1, 2015.

[41] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, Apr 2014.

[42] J. Fan, F. Han, and H. Liu, "Challenges of Big Data Analysis," *National Science Review*, vol. 1, no. 2, 2014.

[43] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *Hawaii Intl. Conference on System Sciences*, 2013.