Bad Big Data Science

Frank S. Haug
Graduate Programs in Software,
University of St. Thomas
St. Paul, MN, U.S.A.
fshaug@stthomas.edu

***Abstract*— As hardware and software technologies have improved, our definition of a "manageable amount of data" has increased in its scope dramatically. The term "big data" can be applied to any of several different projects and technologies sharing the ultimate goal of supporting analysis on these large, heterogeneous, and evolving data sets. The term "data science" refers to the statistical, technical, and domain-specific knowledge required to ensure that the analysis is done properly. Techniques for managing some common causes for bad data and invalid analysis have been used in other areas, such as data warehousing and distributed database. However, big data projects face special challenges when trying to combine big data and data science without producing inaccurate, misleading, or invalid results. This paper discusses potential causes for "bad big data science", focusing primarily on the data quality of the input data, and suggests methods for minimizing them based on techniques originally developed for data warehousing and distributed database projects.**

Keywords*— *Big Data, Data Quality, Data Warehousing, Distributed Database, Metadata.

I. DEFINING BIG DATA SCIENCE QUALITY

Big data systems are large, distributed, and complicated data systems. Data science activities encompass several sophisticated analytical processes. Therefore, any project tasked with implementing a big data system and performing data science activities on it, must include a combination of complex distributed systems and complex analysis. How do we evaluate the quality of such a project? This paper attempts to address a related and more important question: how can we recognize potential quality issues and prevent the quality from being unacceptable, i.e., how can we prevent "bad big data science"?

The product we produce in any big data science project is the combination of the big set of data itself, the set of analysis we perform, and the ultimate decisions we make based on that analysis. If we wish to evaluate and improve the quality of our product, we need to understand what data quality means and how it might become bad quality. Unfortunately, the definition of good quality and bad quality is both easy to understand and hard to define succinctly.

A. The five views of quality

What does product quality mean? In [1], Garvin asks this question and considers the answer from several different perspectives, such as philosophy, economics, marketing and operations management. He presents five views of quality: the transcendental view, product view, user view, manufacturing view, and value view. We can summarize Garvin's five views by giving a short statement of the primary perspective for each. For example, in [1] Garvin uses the quote, "even though quality cannot be defined, you know what it is" to demonstrate the transcendental view. The product view considers quality to be based on the quality of its ingredients (all of the inputs to all of the processes that produce the product). The user view recognizes the subjective nature of quality, i.e. that quality is evaluated based upon how well it satisfies the needs of the product's users. This view also recognizes that, since we have different types of users, our definition of quality must, consider all of their different needs and expectations. The manufacturing view considers quality to be a measure of the production processes, i.e. quality is a measure of the degree to which both the products and production processes themselves conform to their respective specifications. The value view emphasizes that the degree of quality required to be considered "good quality" or "bad quality" is tied to the definition of "acceptable cost". For example, we expect higher quality from a product that costs more, such as the difference in our quality expectations for an "enterprise edition" versus a "student edition" of the same product. See Table 1 for an informal summary of each view's description of "Good Quality".

B. Beyond the five views

Our understanding of quality in general (and our view of software and data quality in particular) has continued to evolve. Others have made significant contributions, often building on Garvin's foundation.

In [2], Kitchenham and Pfleeger apply Garvin's approaches and further explore this question with respect to software quality in particular. They conclude that the five different views are necessary, but can lead to conflicts and disagreements between the various stakeholder groups.

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

TABLE I. SUMMARY OF THE FIVE VIEWS

View name	Informal Description of Each View's Concept of Good Quality
Transcendental	Good Quality is un-definable; it is only recognized by experiencing it.
Product	Good Quality is a function of the quality of the ingredients (inputs to the process).
User	Good Quality is based on each user's needs and (perhaps unspoken) expectations.
Manufacturing	Good Quality is based on conformance to specifications and process.
Value	Good Quality is based on how much the customer is willing to pay for it.

In both [3] and [4], English considers the definition of software quality with a special focus on data and information quality within analytical information systems. In [3], he argues that the movement of a set of data throughout an information system also represents an evolution; the data set is transformed into concepts containing higher degrees of quality, which he calls "information, knowledge, and wisdom". English claims that this evolution is the result of metadata applied at each stage of our data set's journey. In this context, the term "metadata" can be understood as any details surrounding our data sets and processes. These details can pertain to the administration, understanding, or use of any or all of these things surrounded by the metadata. Stakeholders exist at each step along this path. This list of stakeholders includes the initial data modelers and application programmers. It also includes the people entering the data, and the people implementing the next stage in the data set's journey. As we follow the path our data set takes, it leads to the report and query authors, the decision makers who base their decisions on the analysis results, and ultimately, all of the employees, customers, and shareholders who are affected by those decisions.

Whenever these stakeholders encounter the data moving through the system, they are potential customers of the data, potential knowledge workers who strive to assess and improve the data quality (by using and improving the relevant metadata), and potential producers of the next stage in the data evolutionary process / path. English considers several different sets and types of metadata. This includes details pertaining to semantics and significance, the effort and resources used by each step of the transformation and the status for each of the transformation processes). There is important metadata surrounding the usage (e.g. details reflecting timeliness and relevance of set of data with respect to the processes that use that set) and surrounding the people (e.g. the experience level of the people assigning and using the metadata). Each of these metadata sets needs to be captured, maintained, and integrated into the overall journey that our data set makes through the system.

C. Big data science quality

If we combine these different views and contributions, we can consider what our definition of quality should be for big data science. As data sets are processed by our big data projects, the sets of metadata surrounding them are used and enhanced to improve the quality of the subsequent products and processes. The various data sets and metadata sets are combined with the sets of transformations that we apply, and the queries we execute, to produce a set of decisions. These decisions are the ultimate result of all our big data and data science activities. The quality of big data science must therefore include the quality of these ultimate decisions as well as the quality of all the "ingredients" that lead to them. This definition of quality implies that our overall measurement of big data science quality is dependent upon the quality levels measured for each data set. It also depends on the quality levels measured for each piece of relevant metadata, i.e. all the metadata surrounding each and every one of our data science activities that lead to the final decisions. All of these quality levels rely on several quality evaluations, which are performed by different stakeholders, based on different views and criteria, distributed across our big data systems and processes, and applied at different stages (different locations and times within the system work flow and data flow). For example, data stewards would capture details about the source data, such as completeness (e.g. examining null values for attributes and relationships); while data transformation developers would capture details focused on source versus destination aspects (e.g. cross-footing the number of accepted and rejected transformations). Stakeholders in later stages would focus more on pragmatic quality details (such as the applicability and reasonability of the data, or performance aspects such as data latency). Suggestions for developing such an evaluation methodology can be found in [5], along with a good summary of the overall goals we hope to achieve. If we want to produce good quality big data science, then all these quality details need to be captured, communicated, evaluated, maintained, and improved. Therefore, understanding the stakeholders and including there feedback in the quality process is vital. A good example of how stakeholder analysis and feedback can be integrated into the workflow process is discussed in [6].

II. TRADITIONAL AND BIG DATA SYSTEMS

Big data science projects are different from traditional information analysis and data processing projects. However, big data projects can interact with traditional systems, often traditional systems act as the ultimate source for the information being analyzed. Big data science systems also share some aspects in common with different parts of different traditional systems. Therefore, we can apply the product view of quality, and consider traditional data quality issues as "ingredients" that have a direct impact on the quality of these newer big data systems. Understanding how to improve those traditional systems can potentially improve the quality of these ingredients. Similarly, these traditional issues and improvement approaches might provide insight into the new challenges faced by big data science systems.

A. Relational database management systems

A relational database management system (RDBMS) can be used to implement traditional data processing systems. These processing systems use database models designed using tools and techniques that are based on Chen's [7] entity relationship modeling technique and Codd's definition of the relational model [8]. An RDBMS can provide an efficient

platform for applications that need to processes data transactions. Databases are designed using Codd's concepts of normalization, to reduce the risks of data corruption or inaccurate query processing when both read and write operations occur at the same time (race conditions). Modeling techniques based on Chen's approach allow for separation between the requirement details and implementation details for data. For example, the schema for a database typically models cardinality details for relationships between entities using diagram notations. These can be implemented using constraints and indexes within the DBMS, freeing the application from the responsibility of enforcement. Applications written to perform online transaction processing (OLTP) can leverage these techniques and other RDBMS facilities to implement constraint enforcement and indexed access methods. Even after the application has been implemented and deployed, a database administrator can use the RDBMS management facilities to optimize these implementations, and improve performance for those database operations that are most critical to the application or most frequently used by the application. Skilled database administrators, data modelers and developers can use the RDBMS to ensure data integrity is preserved while efficiently processing transactions containing thousands of potentially conflicting operations. Access to schema and data content can be controlled per user or for groups of users. Through various techniques, this control can be defined to a fine level of granularity, even down to the column or row-level. Several different RDBMS platforms provide fallback and recovery facilities to ensure that data loss is minimized, even when power failures or hardware crashes occur. However, the details for these various features and facilities must be defined, implemented, and executed correctly. Definition and implementation quality depend on the skills of the developers, and ultimately on the schedule and budget constraints of the project (i.e. this is highly dependent on value view of quality). Similarly, the execution of these operations can cause performance overhead (e.g. transaction processing can correctly handle race conditions, but can lead to locking bottlenecks or even deadlock situations). Therefore, it is possible that the actual design, implementation, and execution delivered for a given RDBMS project might provide less than the maximum degree of quality that the RDBMS is capable of providing.

B. Distributed database management systems

The proliferation of RDBMS and other data systems combined with the need for an integrated view of data, spanning different departments or applications, lead to the concept of a distributed database management system (DDBMS). A DDBMS can be architected using one of several different alternative approaches. A DDBMS can provide a single, uniform interface to the data contained in any of several, separate, traditional RDBMS databases. Ideally, applications written to access the DDBMS can connect to any data controlled by the data systems integrated by the DDBMS, even data managed inside flat files or non-relational databases. As Rahimi and Haug discuss in [9], there are several DDBMS architectural alternatives, but all have a common goal; they strive to provide some degree of support for several different types of transparency. For example, providing location transparency means that the application does not need to know any network or deployment details for the underlying host system that contains the data set it needs to access for a given query. Fragmentation transparency means that the DDBMS provides an integrated view of the data, even when the underlying schema and content is split into several groups of rows (horizontal fragmentation), groups of columns (vertical fragmentation), or subgroups based on both types of fragmentation (i.e. hybrid fragmentation). Replication transparency enables duplication of tables or fragments for better performance and load balancing of queries, while also ensuring synchronization of data content for write operations.

When a DDBMS is used to unite several separate databases, it is not always an ideal solution. For example, individual OLTP applications, previously designed and implemented to connect directly to the underlying data, still need to perform read and write operations on the underlying data sets. Unfortunately, it is often considered impractical (based on the value view of quality) to rewrite these traditional applications to use the DDBMS instead of the direct access they originally used. Because of situations like this, and other considerations, the use of DDBMS for write operations, can become quite limited or even impossible. In such an environment, performing queries against a DDBMS has potential benefits; however it relies on the ability of the DDBMS developers to integrate the data from the different underlying data systems. The task of integrating these systems can be quite challenging, or even impossible, based on the data quality of these systems and the different and often incompatible understanding of concepts across different applications. Therefore the DDBMS quality is dependent upon the data quality of other systems and processes. This means that the degree of quality a DDBMS can reasonably provide is often less than the theoretical maximum that the DDBMS approach could potentially provide.

C. Data warehousing environments

A different approach, called data warehousing, began its development in parallel with the DDBMS approach. In [10], Inmon describes data warehousing as the culmination of a series of evolutions in data processing and decision support systems. Reference [10] also includes Inmon's claims that in the 1960s, master files and proprietary programs made the data access environment feel "stifling". Summarizing Inmon's view, he states that solutions caused other problems, which in turn inspired solutions with new challenges. For example, poor access to data can lead to extract programs, which can lead to synchronization issues. Solving these issues transformed data into more manageable information and preserved change history, but these transformations could easily lead to "stovepipes", which are situations where different and incompatible attempts to integrate the same data. These transformations could also lead to other issues with data quality and credibility. Ultimately, this lead to the creation of a data warehouse environment, as [10] defines it, this must be a "subject oriented, integrated, non-volatile, and time-variant collection of data in support of management's decisions". Inmon created a data warehouse architecture that initially

focused on a centralized data warehouse, the one, and true source for data integration and capable of supporting extracts in the form of additional data warehouses and smaller subsets called data marts.

Inmon has evolved his approach into a new architecture (defined in [11]) that supports decentralized / distributed data warehousing. But Inmon is not the only voice shaping data warehousing. Kimball, Ross, and Thornthwaite have also been actively working to shape the architecture, lifecycle, and expectations for data warehousing. Kimball's approach, as defined in [12] is different from Inmon's, most notably with respect to centralization versus distribution. Kimball's approach has always been distributed. In Kimball's view, the data warehouse does not physically exist (it is certainly not a centralized database); it is instead a logical union of all the data marts, controlled by a construct he calls the "matrix".

Although Inmon, Kimball, and others have implemented the concept of a data warehouse differently, there is some consensus. In particular, Inmon's four original defining characteristics are generally agreed upon and defined consistently. Subject orientation means that the data warehouse should be modeled using language and concepts that are based on the enterprise subject areas. This means that the business user's view (as opposed to the application or database developer's view) should be captured in the various data warehousing models. Saying that a data warehouse environment is "integrated" refers to the need for combining and cleansing data from many different data models across the organization. The "non-volatile" characteristic means that a data warehouse environment will typically avoid deleting or updating data when it changes (loading data is thought of as an "append new version" operation rather than an update). Lastly, "time-variant" is a somewhat related concept, namely that each value loaded in the data warehouse should have an associated date and / or time.

These characteristics are radically different from traditional online transaction processing (OLTP) models, but so are operations being processed by the transactions and queries. The data warehouse environment is built to perform online analytical processing (OLAP), not OLTP. Data warehouse environments perform separate data modification (loading) from data reading (querying, reporting, mining, etc.). Loading typically occurs as part of a separate process called extraction transformation and loading (ETL), typically as a batch process performed during off-peak hours, and periodically rather than constantly. ETL is metadata driven (based on metadata capturing the source to target mapping details). ETL is also responsible for assessing data quality, attempting to cleanse "dirty data" when possible, and managing all the metadata related to the population processes. Queries are different for OLAP than OLTP, typically focusing on aggregate functions, using cached or partially cached data, and mostly involving historical data that never changes after it is loaded (non-volatile and time-variant). Although some systems do focus somewhat on current data (e.g. tactical systems such as operational data stores or operational data marts), the majority of data stores in the data warehouse environment are more strategic (updated less frequently, and more focused on past values than current values).

D. Big data systems

The term "big data" is relatively new to academia and business, with the current views originating in the seminal "Bigtable" paper [13]. Many academics and business professionals are still learning about big data and exploring its many facets; however consensus about many of its core principles and concepts has begun to emerge. The definition for big data revolves around three concepts (the three V's); "Volume, Velocity, and Variety" [14]. Volume refers to the size of the information itself, the depth and breadth of data about a transaction or any point of interaction (from a data warehousing perspective, these are the facts and dimensional attributes associated with a business process or subject area). Velocity refers to the pace at which this information is generated (e.g. according to Jay Parikh, Facebook's vice president of infrastructure engineering, Facebook ingests 500 TB data per day [15]). Variety refers to the "incompatible data formats, non-aligned data structures and incompatible semantics (e.g. web click data, tweets, non-standard URLs, etc.)" that form a "barrier" to creating a single, static schema for data [14]. In data warehousing, this concept is sometimes referred to as semantic impedance mismatch. This "mismatch" or "barrier" is a situation that both Kimball and Inmon have attempted to minimize or avoid with their methodologies. Big data projects need to embrace the situation and address the issues more directly than data warehousing environments.

Current trends for processing big data are built around an infrastructure of several interconnected open source projects (often referred to as an ecosystem) and a potentially large number of commodity-hardware machines (usually referred to as a cluster or data center). Clusters can include tens, hundreds, or thousands of machines, data centers can contain multiple clusters, and organizations can conceivably contain multiple data centers. The commodity hardware machines that make up these larger concepts are typically multi-core machines with several gigabyte of ram and several terabyte of disk space. Although these machines are not slow or powerless, they are quite small compared to the large enterprise servers traditionally tasked with more centralized approaches. Similarly, although these machines do not have worse-than-average reliability, the architecture expects failures to occur (simply due to the large number of machines and realistic expectations for mean-time-to-failure). Therefore, the infrastructure embraces the ideas of massive parallelism (for performance) and reasonable redundancy (for performance and fault tolerance) at a reasonable cost.

III. BIG DATA SCIENCE QUALITY ISSUES

Like projects in a data warehousing environment, big data science projects work with data that might have been loaded from several different and dirty sources. It is possible that the data might be merged into an integrated format for a big data project, but this not necessarily required. Many big data technologies, such as MongoDB [16] support self-defining schemas also called schema-less databases. The term "self-defining schema" means that there are schema details (metadata) embedded in the file format, usually on a per-record basis. This means that each record can represent more than merely another row in a table with a fixed schema, for

example, it also enables columnar data formats. This also enables each new record to potentially change the schema definition for that record. For example, an Extensible Markup Language (XML) file contains the names of the entities and attributes as well as the content for each entry in the file. This enables greater flexibility for the source file formats, however, this also can mean that integration is more complicated than traditional data warehousing or distributed database integration. It is possible to use these formats for both the source data storage and the target data storage. Because the schema is self-describing on a record-by-record basis, it is essentially schema-less (capable of changing dynamically as the file is processed). This can also mean that the integrated target schema has little if any visibility outside the program code.

A. Duplication, Fragmentation, and Location Issues

Within a given big data science project, there can be several sets of data. Similar to data warehouse environments, some of these sets might be capturing changes over time within the same data source (e.g. once a day, after the close of business the daily sales data could be extracted from the traditional OLTP point of sale system and loaded into a separate target data set for each day). While extraction programs could attempt to minimize duplicating unchanged data across the loads, because the big data technologies do not behave like traditional RDBMS data sets, this can be time consuming and difficult to do with the big data technologies.

In addition to potential duplication issues, across different big data science projects, it is possible that the same data sets might be extracted from (as a source data set) more than once. It is also possible that the same source might be transformed and loaded into more than one target data set, possibly using different programs, different schemas, and different transformations. These different resulting data sets potentially contain different pieces of the original data set. Similar to a distributed database system, the data has potentially been fragmented into different pieces, but unlike a DDBMS, there little if any visible documentation of the fragmentation details. As discussed in [9], a DDBMS would ensure that the fragments can be put back together without gaps or overlaps using one or more "reconstruction programs / plans". But a big data system does not necessarily have any such guarantee. If these source or target data sets are shared with multiple big data science projects, it can result in a confusing collection of very large, overlapping, interdependent data sets (similar to the issues that Inmon described as leading to data warehousing). Even if there are no reconstruction issues, there certainly is no support for fragmentation transparency; if a program queries the wrong attributes for a given data set, it will simply retrieve no values for those attributes, even if a different data set (fragment) might contain the requested attributes for the same source data records.

Lastly, even if duplication and fragmentation issues are kept to a minimum, there is no standard mechanism for finding the correct semantic description, administrative details (e.g. date and time the extract occurred), or naming details (e.g. path and filename location) for a given set of data. In distributed database terms, this means that there is little if any support for location transparency. It is not difficult to specify the data set naming details as part of the run-time or deployment details for a big data science program, but this is not typically tied to the semantic or administrative metadata details mentioned. These naming details are also often hard-coded into the command line or shell script used to control the execution; they are not typically provided in a more metadata-driven fashion.

B. Stovepipes and Integration Issues

Suppose two different departments (e.g. Sales and Marketing), within the same enterprise, each decided to build their own data projects. Further suppose that the Sales group defined the concept of a Customer as representing "anyone who has every bought our product", and defined attributes and relationships based upon their existing OLTP systems. Similarly, the Marketing group has defined Customer based on their perspective, as representing "anyone who has ever expressed interest in buying our product", and containing the attributes and relationships that their OLTP systems supported. As an organization, we would have a serious consistency issue. Simple analysis across the enterprise now suffers because we cannot obtain a single, consistent version of the truth (for example, the question "How many customers do we have?" will receive a different answer depending upon which data set we query). The lack of communication (or building the data sets in isolation) is referred to as a "stovepipe", and it is a serious issue for any cross-department or enterprise-wide analysis. The term "stovepipes" reflects the fact that other groups might choose to build off of one of these isolated groups, leading to several trees or graphs being built across the organization, supporting dependencies within each tree / graph but not across them.

Data warehouse projects contain lifecycle processes and architectural components (such as the central data warehouse in [10] and [11], or the "matrix" in [12]), which prevent stovepipes from occurring. Similarly, distributed database projects have processes and components to prevent or minimize stovepipe situations ([9] emphasizes the importance of the global conceptual schema, which is an integrated schema created my merging all the local schemas being combined across the enterprise by the DDBMS). Because big data science projects do not necessarily have a lifecycle model process or architectural component that enforces consistent integration across the enterprise, a situation similar to the "stovepipes" can easily occur across big data projects. Even when the situation is not as drastic as the previous example, issues with integration can be hidden (due to the lack of communication and coordination across the sets). When this is combined with the location and duplication issues, the quality of the analysis can silently suffer.

C. Additivity Issues for Multidimensional Analysis

When analysis is performed on a set of data, we often use formulas and functions that process a large number of data values as input, while producing a lesser number of data values as output. For example, we might produce a single output value representing the sum of hundreds, thousands or even millions of individual input values. Similarly, we could perform a different operation on the same set (e.g. average,

count, minimum value, or maximum value) to produce a single output based on numerous input values. In data warehouse projects, these formulas and their output values are called aggregates.

Ideally, the input values we feed to these aggregate formulas can be broken down further, i.e. rather than processing a large number of raw input values directly, we can often partition the input values, forming multiple, mutually exclusive subgroups of the original input set. Then, instead of processing a single data set with a large number of individual elements, we can process several input data subsets, each containing a smaller number of elements than the original set.

Often, we can repeat this partitioning process several times on the subgroups. This results in a grouping of input values along a classification hierarchy of contextual values. These contextual values surround the data being aggregated. This is called dicing the data. For example, suppose we calculated the total sales amount (one aggregate) calculated across all the individual sales amounts collected within the entire country (one set containing all the sales amounts in the entire country); here we have diced the data by country. We could then break this analysis down again, by calculating several aggregates, grouping the sales according to the particular state within the country where the sale occurred (e.g. calculating fifty total sales amounts, one for each of the fifty states contained within the country). This new example would be dicing the data by country and then by state, rather than simply dicing by country. Starting from a large category and recalculating for a subset within the same hierarchy is called "drilling down". If we did the opposite (e.g. starting from the fifty aggregates, calculate the grand total for the entire country instead) we call it "rolling up".

We could also selectively include or exclude some of the data values from these aggregate operations (e.g. excluding the sales of a particular brand of product, while still dicing by country and state); this is called "slicing". We can perform these operations across different category hierarchies (called dimensions) at the same time, resulting in activities known as "slicing and dicing" the data. This is a common and powerful technique for analyzing the data. For example, in our analysis, we could recalculate the total sales amount aggregate, broken down by year, month or day, and then by county, city, or state. This is called multidimensional analysis. It is the primary purpose behind data warehousing, and why data is modeled dimensionally within the data warehouse environment. Switching between different slice and dice operations can be easily implemented and visualized using a pivot table.

As part of the dimensional modeling process, we identify each piece of data (called a fact) that might be aggregated versus each contextual detail (called a dimension attribute) that surrounds the fact. We also determine whether there are any potential quality issues with the set of all possible slice and dice operations on a given set of facts and hierarchies of dimensional attributes. In the dimensional model, we capture this information as metadata called the "additivity type", which is defined for each fact in the model.

As explained in [18], a fact is additive for a given dimension "if the sum operator can be used to meaningfully aggregate values along all hierarchies in that dimension." Each fact can be specified as fully-additive, semi-additive, or non-additive, based on whether the fact is additive for all, some, or none of the dimensions surrounding it (respectively). For example, a bank account balance is not additive across time. Suppose we had one hundred dollars in the account yesterday. Further suppose that we still have one hundred dollars in the account today. If we summed the account balance across the two days, we would have a total of two hundred dollars, but that is not the total amount of dollars in the account! Similarly, some facts can be non-additive, e.g. values that are not really numbers (e.g. zip-codes, phone numbers, dates). Ratios provide a more subtle example of non-additive facts. For example, suppose we recorded the "percentage full" for two fuel tanks of different sizes (e.g. 10 gallon versus 100 gallon). These ratios cannot be simply added, we would need to capture the numerator and denominator values separately and perform the correct mathematical function if we wanted to aggregate the percentage full.

Big data science systems can also perform multidimensional analysis; however the data is typically not modeled dimensionally. This means that facts and dimension hierarchies are not modeled and additivity is not specified for the facts. Therefore it is possible for some piece of analysis within our overall analysis process to perform operations that are meaningless or misleading (such as adding the percentages or adding values across invalid groupings like the bank account example). Based on the product view, we can argue that the quality of the subsequent analysis and the quality of the ultimate decisions made are damaged when we have bad quality due to additivity issues.

D. Lineage and Version History Issues

Unlike traditional OLTP systems and DDBMS systems, big data systems, tend to be non-volatile, similar to data warehousing environments. This means that data modifications are captured in big data projects as a new version of the previous record, typically appended to the same data set or added to some newer data set (i.e. the old value is not updated in place or removed). Unlike a data warehouse environment model, there is no lifecycle process or architectural component enforcing the time-variant property (data warehouses always include some form of date or time dimension). Worse than that, there is no guarantee that the new version will be distinguishable from the old version (i.e. aside from the changed attribute values, we might not be able to recognize if this is a new version of an old record, a completely new record, or a corrupted and duplicate version of the old record). If it is not handled appropriately, this situation could have dire consequences for the analysis. As an extreme example, we might not be able to determine if a set of data contains 10,000 individual customers with one version each or if it contains 5,000 customers with 2 versions for each. Real data scenarios would contain many more records, versions, and complex combinations of the possible situations.

If any of the analysis produces surprising or unexpected results, our analysts might reasonably want to know where the results came from. If they subscribe to English's description of data movement and evolution (described in [3] and [4]), they

would want to trace all the transformations backward to the original data, and potentially examine the surrounding data and metadata at each stage along the way. In data warehousing, this is called tracing the data lineage, and it can be supported using standard data warehousing constructs and techniques. However, data lineage is not necessarily captured in big data projects, and not directly represented in the analysis results, data sets, or associated metadata. This means that questionable or unexpected results can be difficult or impractical to verify and validate.

IV. STRATEGIES FOR POTENTIAL SOLUTIONS

If one or more of the issues mentioned in the previous sections are present among our collection of big data science projects, how can we address these issues? The primary cause for the issues mentioned is lack of metadata management integration. There are other considerations necessary, but the lack of metadata capture and visibility combined with the three V's make it very difficult to manage the complexities that can lead to bad big data science quality. In general, the solution to each of the issues mentioned will require a combination of architectural components, underlying technology, and human software processes. The issues identified in previous section followed a natural order for discussing the issues. The strategies for addressing those issues will be presented an order based on the recommended implementation order. The strategies will consider the issues in this different configuration and order to allow the strategies to build incrementally upon one another chronologically.

A. Addressing Location Issues

A simple metadata catalog could be implemented in a small database to capture data set location and semantic details. Although the size and complexity of our big data sets is too much for a RDBMS to handle, the set of names, purposes, and file locations for these data sets is easily manageable. While it is not strictly necessary to use a relational database, the RDBMS technology can provide application independence, integrity, concurrency, durability and fallback facilities for the metadata needed to capture location details and support location transparency. A simple set of database operations can replace the hard-coded parameters used by our programs or shell scripts.

B. Addressing Fragmentation Issues

Fragmentation issues are more complicated. The variety of our data set schema (schema-less and self-describing schemas) make this more difficult than traditional systems. However, we could begin by creating a metadata schema capable of capturing and documenting the fragmentation details. This would be similar to the fragmentation details captured in the global conceptual schema discussed in [9], but it would need to be more flexible and adaptable. In particular, it might need to be populated during the development of the data extraction and loading programs, based on the actual data (and schema) encountered. Ideally, this would be populated during development and used to drive the programs; but initially we could simply populate it based on the existing programs. This fragmentation metadata would need to connect to metadata from the source systems as well as the location metadata mentioned in the previous section. Implicitly that means that we must also define a schema capable of capturing the data dictionary details from the source systems (or the equivalent metadata information). If we have populated the source schema, destination location schema, and fragmentation schema, we have created something similar in its function to parts of both a DDBMS global conceptual schema and a data warehouse transformation and mapping metadata management system.

C. Addressing Duplication / Replication Issues

Once the data set location and fragmentation metadata schemas have been defined, we can extend them to support capture of replication details. I am using the term duplication to refer to situations where two or more data sets contain the same content and schema, but not through any intentional or coordinated effort. By contrast, I will use the term replication to refer to this planned and coordinated redundancy. By intentional replication, I am claiming that we can ensure synchronization between the replicas; unintentional duplication might be synchronized or not synchronized (since it is unplanned and uncoordinated by definition).

Ideally, we would evolve the system to replace all duplicates with replicas, plan and populate the replication metadata from the top down, and use it to implement metadata driven control of our processes. However, initially, we might need to manually define and populate duplication metadata based on details reverse engineered our existing processes and data sets. We could then manually generate the replication details based on analysis of the duplication metadata. This initial population does not initially provide a direct benefit to the applications involved, but it can help the developers and administrators to evaluate the existing data set quality, identify potential issues, and possibly identify opportunities for future reuse among the data sets and processes.

If we had not done so earlier, now would be a good time to consider the definition and population of a metadata schema for capturing process dependencies and other associated metadata. We could then consider implementing synchronization controls based on replication and duplication details. Population of this metadata could be integrated into post-processing for our existing programs, or implemented as a separate task based on the software and script designs and implementations. This metadata could then be expanded to include performance and tuning details to support more sophisticated optimization of out processes.

D. Addressing Stovepipes and Integration Issues

Stovepipes and integration issues occur when we do not have sufficient capture, visibility and control over the mapping details for our data sets. This lack of control and vision can potentially lead to isolated sets of data with incompatible understandings and transformations for our data. If we have implemented the location, fragmentation, and replication control mentioned in the previous sections, we can consider analyzing the fragmentation details and identifying the potential conflicts. This does not solve the integration issues, but it does allow us to recognize when they are present and

when they are resolved. Similar to the replication discussion, for each integration conflict, we would need to manually reverse engineer the correct integration plan, implement the processes, convert the data, and capture the details to ensure that the issue is resolved. While we can augment the fragmentation schema to include support for a top-down definition of fragmentation (analogous to the reconstruction program from [9]), we would need to manually define and implement these changes. If we have a large number of data sets, and we have existing fragmentation with a significant amount of gaps and overlaps, this can require a large effort to resolve.

E. Addressing Additivity Issues for Multidimensional Analysis

Additivity can be an extremely subtle concept, even within a data warehouse environment. It relies on an understanding of the underlying scale type (see [19]) for the data, but also more subtle understanding of the semantics of the data being aggregated. In data warehousing, the data being aggregated are called facts, the contextual data surrounding the facts are called dimensions, and they are organized in structures called dimension hierarchies. Data quality issues can also destroy additivity (e.g. duplicate and missing values for a hierarchical value can cause bad analysis). Even if we have successfully addressed the issues from the previous sections, we might not be able to completely address additivity issues, but we could potentially try to capture dimension hierarchies among the attributes in our big data sets. If we have already addressed the stovepipe issues, we might be able to define dimension hierarchies based on the agreed upon definition for all the attributes (after removing the integration issues). This could be captured in a simple metadata schema, similar to the metadata storage used in a data warehouse environment for dimension hierarchy and fact additivity details. Solving additivity issues still relies on manual population of the metadata, and manual evaluation of queries that potentially violate the additivity. Therefore, this approach is not really tailored to the big data environment, and is not guaranteed to work for all projects or data sets.

F. Addressing Lineage and Version History Issues

Implementing the fragmentation and replication metadata in the previous sections can be used as a foundation for storing lineage metadata, at least within the big data sets. Lineage metadata for the data systems used as a sources are not necessarily present. Therefore, just like a data warehouse environment, we might need to accept imperfect lineage details, or improve the production systems to keep track of these metadata details. Providing support for proper version history is analogous to data warehousing's slowly changing dimensions strategies, as discussed in [10], [11], and [12]. Implementing these strategies in combination with support for additivity and multidimensional analysis would require each data set to include additional keys for the hierarchy levels (to support proper population of the versions within a hierarchical dimensional representation). The more extensive change would be the additional processing and adjustments that would need to be added to the extraction, cleansing, loading, and analysis processes. These changes would be needed to ensure that multiple versions do not cause incorrect analysis (e.g. preventing improperly identifying two versions of one customer as two customers).

V. CONCLUSIONS AND FUTURE DIRECTIONS

Not all quality issues can be addressed within the scope of a single paper, metadata model, or process lifecycle model. Big data science projects are large distributed systems with complex data sets and sophisticated analysis processing. In this paper, the suggested strategies for addressing potential quality issues would result in additional complexity, but hopefully the resulting quality would be improved by these changes. It is possible that these changes might ultimately lead to some simplifications in other areas.

A. Summary

This paper considered the potential big data science quality issues, by focusing on quality issues with the input data sets and the processes used to transform these distributed, autonomous sets into an integrated big data set. The strategies presented here began by considering traditional quality concepts and examining how similar quality issues have been addressed in relational database, distributed database, and data warehousing environments. Big data science quality is a sophisticated concept to measure. It involves many data sets, metadata sets, processes, stages, people, and points of view. We can evolve our ability to measure quality by evolving our system to prevent some of the potential causes of bad big data science quality. The primary causes behind the issues identified in this paper included the underlying complexity of the data sets and analysis themselves, the potentially uncoordinated data management and software development processes, the minimal visibility for potential problems, and the lack of metadata integration and management. Solution strategies were suggested based on incrementally adding metadata capture, population, and management to the data sets and software systems. These strategies also require modifications to the human processes used to develop and manage the data, metadata, and programs for these projects.

B. Future Directions

A more detailed design of the metadata models discussed in the strategies has not been created yet. Tools to enable better metadata capture and metadata driven extraction, transformation, loading, and analysis has not been explored. Some existing data warehousing tools can connect to big data sets, but they have not been designed to address the big data issues described in this paper. These strategies have not been tried on any real world big data science projects. All of these would be potential starting points for additional research.

REFERENCES

[1] D. Garvin. What does "product quality" really mean? Sloan Management Review (Pre-1986) 26(1), pp. 25. 1984.

[2] Kitchenham, B. and S. L. Pfleeger. "Software Quality: The Elusive Target Special Issues Section]." Software, IEEE 13 (1): pp.12-21. doi:10.1109/52.476281.

[3] L. P. English. Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits 1999.

[4] L. P. English. Information Quality Applied: Best Practices for Improving Business Information, Processes and Systems 2009.

[5] J. S. Saltz. The need for new processes, methodologies and tools to support big data teams and improve big data project effectiveness. Presented at Proceedings of the 2015 IEEE International Conference on Big Data (Big Data). 2015, Available: http://dx.doi.org/10.1109/BigData.2015.7363988. DOI: 10.1109/BigData.2015.7363988.

[6] J. S. Saltz and I. Shamshurin, "Exploring the process of doing data science via an ethnographic study of a media advertising company," 2015 IEEE International Conference on Big Data (Big Data). Institute of Electrical and Electronics Engineers (IEEE), Oct-2015.

[7] P. Chen Pin-shan. The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems (TODS) 1(1), pp. 9-36. 1976.. DOI: 10.1145/320434.320440.

[8] E. Codd. A relational model of data for large shared data banks. Communications of the ACM 13(6), pp. 377-387. 1970. DOI: 10.1145/362384.362685.

[9] S. Rahimi, F. S. Haug and I. C. Society. Distributed Database Management Systems: A Practical Approach (1st ed.) 2010.

[10] W. H. Inmon. Building the Data Warehouse (4th ed.) 2005.

[11] W. H. Inmon, William, D. Strauss, and G. Neushloss, DW 2.0: The Architecture for the Next Generation of Data Warehousing 2008.

[12] R. Kimball, M. Ross, and W. Thornthwaite, The Data Warehouse Lifecycle Toolkit (2nd ed.), 2008.

[13] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes and R. E. Gruber. Bigtable: A distributed storage system for structured data. Presented at In Proceedings of the 7th Conference on Usenix Symposium on Operating Systems Design and Implementation - Volume 7. 2006.

[14] D. Laney "3D data management: controlling data volume, velocity and variety." Internet: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf, Feb. 6, 2001, [Sep. 29, 2016].

[15] D. Tam, "Facebook processes more than 500 TB of data daily." Internet: https://www.cnet.com/news/facebook-processes-more-than-500-tb-of-data-daily/, Aug. 22, 2012, [Sep. 29, 2016].

[16] H. Abbes and F. Gargouri. Big data integration: A MongoDB database and modular ontologies based approach. Procedia Computer Science 96, pp. 446-455. 2016. DOI: http://dx.doi.org/10.1016/j.procs.2016.08.099.

[17] O. Hajoui, R. Dehbi, M. Talea and Z. Batouta. An advanced comparative study of the most promising nosql and newsql databases with a multi-criteria analysis method. Journal of Theoretical and Applied Information Technology 81(3), pp. 579-588. 2015.

[18] J. Horner, I. Song and P. Chen P. An analysis of additivity in OLAP systems. Dolap '04 pp. 83-91. 2004. DOI: 10.1145/1031763.1031779.

[19] S. S. Stevens. On the theory of scales of measurement. Science 103(2684), pp. 677-680. 1946. Internet: http://www.jstor.org/stable/1671815..