Reference Architecture and Cost Estimation Model for Building Intelligent Platforms

by

Mohammad Jouni

B.S., American University of Beirut (2007)

Submitted to the System Design & Management Program
in partial fulfillment of the requirements for the degree of

Master of Science in Engineering & Management

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Signature redacted

Author ....
Mohammad Jouni
System Design & Management Program
August 11, 2017

Signature redacted

Certified by... ....
Bruce G. Cameron
Director, System Architecture Lab
Thesis Supervisor

Signature redacted

Accepted by ....
Joan Rubin
Executive Director, System Design & Management Program

Reference Architecture and Cost Estimation Model for Building Intelligent Platforms

by

Mohammad Jouni

Submitted to the System Design & Management Program
on August 11, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Engineering & Management

Abstract

With the recent resurgence of Machine Learning and Artificial Intelligence as a competitive advantage in product development, technical executives and managers are interested in learning what it would take to build intelligent platforms that can leverage these advances. In addition, they wish to produce cost estimates for developing such platforms. The goal of this thesis is to develop a reference architecture for an intelligent platform and an associated costing model that allows technical managers to understand the components needed to deliver such a platform and estimate the cost of each module, estimate the cost of the overall architecture, and enable what-if analysis to understand the cost tradeoffs. The intent is not to provide the values of the variables in the model, but to develop a cost model that will enable interested parties to plug in their estimated values for each factor and generate a forecast of the build cost.

Thesis Supervisor: Bruce G. Cameron
Title: Director, System Architecture Lab

Acknowledgments

I would like to thank my thesis advisor, Prof. Bruce Cameron. Thinking back on the whole process of writing this Thesis, I cannot imagine being able to deliver it without his help, support and patience. Prof. Bruce helped me navigate the process while I was holding a full-time job that required constant international travel. He was patient and guiding throughout the process. He gave me enough room to put my own touch on this work, while making sure it delivered substance. A sincere thank you.

To my colleagues at Ernst & Young, thank you for all the support and tolerance throughout this process. Specifically, I would like to thank Ali Khan and Greg Raimann. Without their mentorship and help, this work would not have materialized.

To my sister, Aya Jouni, thank you for all the help. Especially for adding your creative touch to the illustration of the reference architecture.

To my parents, Nada Chemaitilli and Ahmad Jouni, thank you for the countless sacrifices you made. I will never be able to repay you, but I hope this makes you proud.

Finally to my wife, Farah. For the past 2 years at MIT you have cheered me through and made sure I had everything needed to succeed. I could not have asked for a better partner in life. For everything you do that I see, and for the things I don't, thank you. We can now plan that vacation...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With Machine Learning and Artificial Intelligence thrusted back into the technology limelight by big corporations like Google, Amazon and Facebook, technology executives are interested in understanding how they can build the same level of intelligence and capability into their offerings. These executives will find a large number of published work related to developing machine learning models and all the work surrounding the tools needed to develop such models. However, an often missed critical component that these executives should be considering is the overall platform that they will need to build to capitalize on these advances. In this thesis, we will be addressing this gap.

1.1 Motivation

In November 2016, Google released a new version of their translation model that vastly increased the accuracy of their translations. Post update, users found it difficult to differentiate between the translations done by a human and the ones done by Google Translate. This has wide impact on Google's products because the same model will be used by Android users, Google Translate, and Google Search, and anywhere translation is needed in Google's products. This is only possible because Google has a platform that enables it to propagate the technical advances it achieves to the products in its stack. This capability is not isolated to Google, and we believe it is accessible to any engineering organization that employs a platform mentality.

1.2 General Objectives

We hope to empower engineering organizations by giving them a reference architecture that will enable them to build an intelligent platform. We believe this platform will power their journey towards integrating Machine Learning and Artificial Intelligence into their software development capability. We believe that by providing the blueprint for the platform and enabling the engineering team to assess how much investment they will need, we would make it easier to launch these efforts, thus contributing to the advancement of AI and ML in products we use everyday and increasing its ubiquity.

1.3 Literature Review

To successfully achieve the objectives of this thesis, we will perform a literature review that spans the domains of big data, system architecture, and software estimation.

1.3.1 Big Data

To understand the various components needed in an intelligent platform, we will be reviewing published work on big data platforms. In our opinion, big data platforms in literature differ from intelligent platforms by neglecting the data ingestion, AI/ML model management and the exposure of generated insights for consumption by external systems. That said, published work will be instrumental in identifying the core components needed in an intelligent platform.

1.3.2 System Architecture

To develop a usable reference architecture, we need to present the components in a manner that can be used by readers. This entails understanding what a reference architecture is, and how to present it to maximize impact. To achieve this, we will need to review published reference architectures and identify how to communicate our blueprint of the intelligent platform.

1.3.3 Software Cost Models

As we develop the cost model for the reference architecture, we will need to identify all relevant software estimation tools and understand their strengths and weaknesses as they relate to our use case. We will also need to research frameworks that allow us to compare software estimation models to formulate our selection criteria.

1.4 Specific Objectives

We believe we will be able to achieve the goals of this thesis by organizing the work into three core objectives :

1. Compare and critique published data analytics platforms. This will allow us to extract the capabilities needed in an intelligent platform and identify the architectural components that will be needed.

2. Develop a reference architecture based on the reviewed data analytics platforms and any pertinent literature review. This will allow us to create a blueprint for the development of intelligent platforms.

3. Create a cost model for the reference architecture. This will be done by first identifying the ideal software estimation methodology to use, then adapting it for the reference architecture.

1.5 Thesis Overview

The thesis will be organized as follows :

- **Chapter 2**: This chapter will provide an overview of the technical advances that contributed to the demand for intelligent platforms.
- **Chapter 3**: This chapter will present and critique published big data platforms.
- **Chapter 4**: This chapter will present our reference architecture for intelligent platforms. It will expand on the role of each component in the platform and list their interdependencies.
- **Chapter 5**: This chapter will assess and select a software estimation methodologies for the purpose of creating a cost estimation model for the reference architecture presented in Chapter 4. It will present the created cost estimation model in a manner that can be leveraged by any organization that wishes to build the reference architecture.
- **Chapter 6**: The final chapter will list the key accomplishments of this thesis, the limitations and potential followup improvements.

Chapter 2

Intelligent Platforms

2.1 Introduction

The amount of data created annually is projected to reach 163 zettabytes (163 trillion gigabytes) by 2025, up from 4.4 zettabytes in 2013 (Reinsel et al., 2017). This 37x expansion over 12 years – harnessed effectively by companies and the public sectors – has the potential of positively improving our quality of life.

As we collect more data, we are able to unlock new insights and build product features that were not previously possible. Let's consider diabetic patients for example. If we have enough data about their daily activities, glucose level throughout the day and their food intake, we can correlate any spikes and dips in their glycemic level with specific events. We can then tailor activities and diets based on individual patients lifestyle and propensities. Another example can be taken from the banking sector. If we have enough information about a person's spending habits, life stages (Are they looking for a house? Planning a vacation?), and preferences, we can build applications that help a customer save more money, plan better for their retirement or pay their loans faster.

The benefits reaped from applying analytics to large data sets is by no means

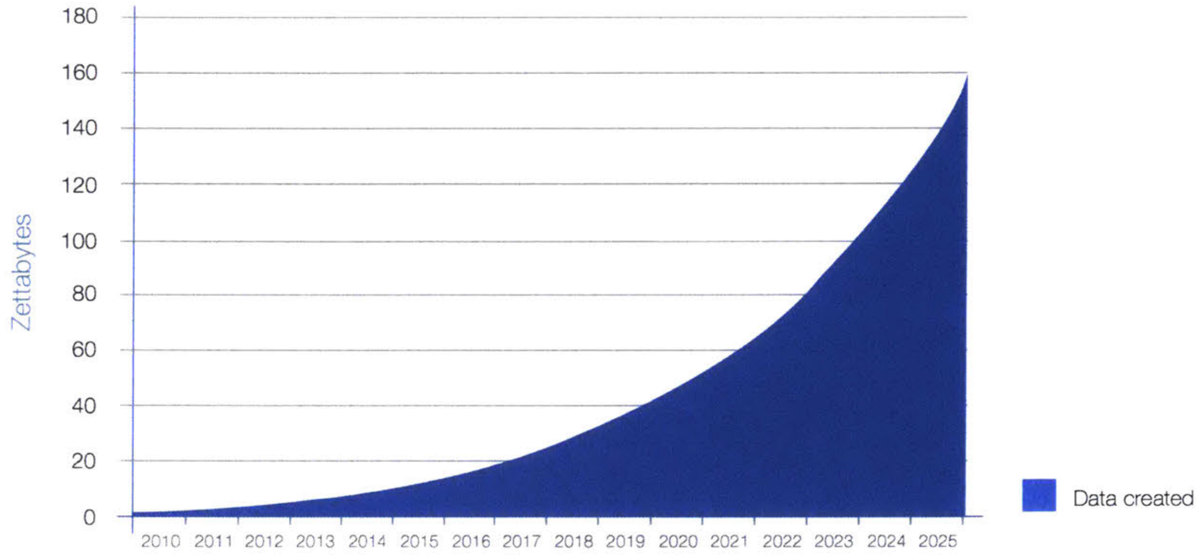


Figure 2-1: Data Growth. Source: IDC's Data Age 2025 study, sponsored by Seagate, April 2017

isolated to a specific industry: A 2013 McKinsey study that surveyed 714 companies representing various industries and company sizes, showed that companies that invested heavily in analytics reaped a 2x return on investment in 5 years. This is an ROI that surpasses returns on R&D and Marketing expenditures (Bughin, 2016).

2.1.1 What Are Intelligent Platforms?

To unlock the full potential of the data collected, companies and the public sector need to invest in the talent and tools needed to enable these analytical efforts. They need to shift their work from one off analytical projects, to building analytical platforms that simplify collection of large datasets of data and convert it into useful insight that can be consumed by other services and applications. We call these platforms Intelligent Platforms.

Intelligent platforms aim to combine the technological advances made in big data processing, cloud computing and AI to enable teams to build intelligent applications. These applications can integrate new use cases that would not have been possible due to cost or lack of needed data. Building these platforms would not have been possible if the concepts discussed in this chapter were not at their current level of maturity.

2.2 Technical Advances In Data Processing

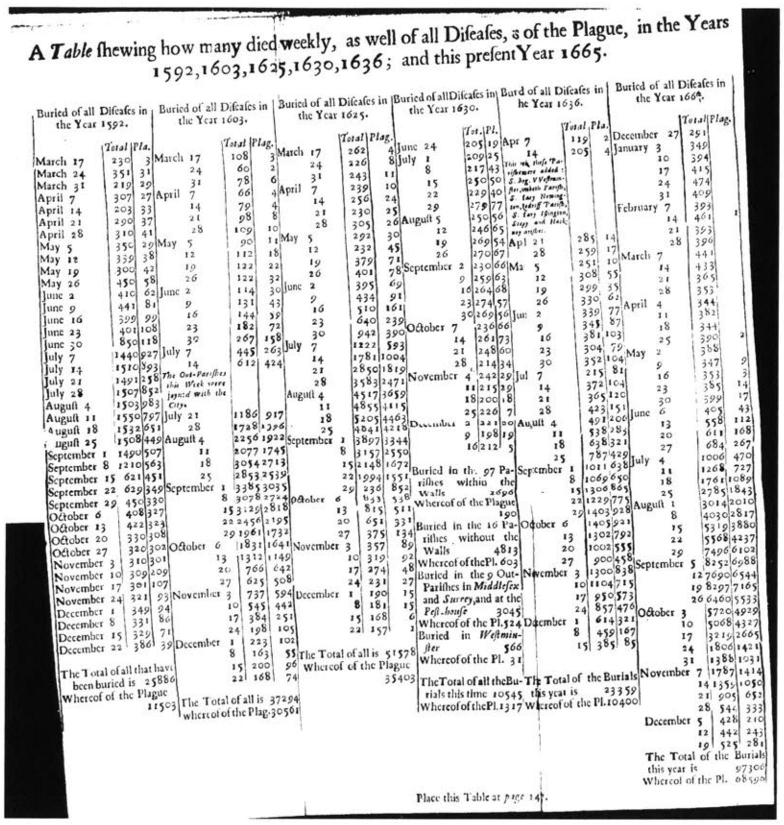
To understand what intelligent platforms are, we need to first look at the history of analytical efforts and the technological advances that enabled each evolution in our analytical capabilities.

2.2.1 The First Statistician

One of the first recorded instance of extracting insight from data is attributed to John Graunt (1620-1674) (Lewin, 2004). In 1661, he used mortality bills that listed the total number of weekly deaths, cause of death, births and marriages to detect and predict epidemic outbreaks. His work helped Londoners decide whether to leave or stay in London based on the expected epidemic outbreaks (Anonymous, 1865). He also leveraged this data to construct life tables which measured life expectancy. He published the results of his analysis in a book titled *Natural and Political Observations Made upon the Bills of Mortality*, which ran for five editions until 1676.

Graunt's work was only possible because Lord Thomas Cromwell (1485-1540) issued a decree in 1538 ordering parishes to record all christenings, weddings and burials (Dennis J, 2016). This data was made available to the public on a weekly basis. The original data was by no means perfect and was missing important dimensional data. For example the data did not include the age at the time of death. Nevertheless, Gaunt's analytical work had great impact and he is considered the father of statistics.

These events shows that for an analytical effort to be successful, we need someone



A *Table* ſhewing how many died weekly, as well of all Diſeaſes, as of the Plague, in the Years 1592,1603,1625,1630,1636; and this preſent Year 1665.

Buried of all Diſeaſes in the Year 1592.

	Total	*Pla.*
March 17	230	3
March 24	351	31
March 31	219	29
April 7	307	27
April 14	203	33
April 21	290	37
April 28	310	41
May 5	350	29
May 12	339	38
May 19	300	42
May 26	450	58
June 2	410	62
June 9	441	81
June 16	399	99
June 23	401	108
June 30	850	118
July 7	1440	927
July 14	1510	893
July 21	1491	258
July 28	1507	852
Auguſt 4	1503	983
Auguſt 11	1550	797
Auguſt 18	1532	651
uguſt 25	1508	449
September 1	1490	507
September 8	1210	563
September 15	621	451
September 22	629	349
September 29	450	330
October 6	408	327
October 13	422	323
October 20	330	308
October 27	320	302
November 3	310	301
November 10	309	209
November 17	301	107
November 24	321	93
December 1	349	94
December 8	331	86
December 15	329	71
December 22	386	39

The Total of all that have been buried is 25886
Whereof of the Plague 11503

Buried of all Diſeaſes in the Year 1603.

	Total	*Plag.*
March 17	108	3
24	60	2
31	78	6
April 7	66	4
14	79	4
21	98	8
28	109	10
May 5	90	11
12	112	18
19	122	22
26	122	32
June 2	114	30
9	131	43
16	144	59
23	182	72
30	267	158
July 7	445	263
14	612	424
The Out-Pariſhes this Week were joyned with the City.		
July 21	1186	917
28	1728	1396
Auguſt 4	2256	1922
11	2077	1745
18	3054	2713
25	2853	2539
September 1	3385	3035
8	3078	2724
15	3129	2818
22	2456	2195
29	1961	1732
October 6	1831	1641
13	1312	1149
20	766	642
27	625	508
November 3	737	594
10	545	442
17	384	251
24	198	105
December 1	223	102
8	163	55
15	200	96
22	168	74

The Total of all is 37294
whereof of the Plag. 30561

Buried of all Diſeaſes in the Year 1625.

	Total	*Plag.*
March 17	262	4
24	226	8
31	243	11
April 7	239	10
14	256	24
21	230	25
28	305	26
May 5	292	30
12	232	45
19	379	71
26	401	78
June 2	395	69
9	434	91
16	510	161
23	640	239
30	942	390
July 7	1222	593
14	1781	1004
21	2850	1819
28	3583	2471
Auguſt 4	4517	3659
11	4855	4115
18	5205	4463
25	4841	4218
September 1	3897	3344
8	3157	2550
15	2148	1672
22	1994	1551
29	236	852
October 6	633	538
13	815	511
20	651	331
27	375	134
November 3	357	89
10	319	92
17	274	48
24	231	27
December 1	190	15
8	181	15
15	168	6
22	157	1

The Total of all is 51578
Whereof of the Plague 35403

Buried of all Diſeaſes in the Year 1630.

	Tot.	*Pl.*
June 24	205	19
July 1	209	25
8	217	43
15	250	50
22	229	40
29	279	77
Auguſt 5	250	56
12	246	65
19	269	54
26	270	67
September 2	230	66
9	259	63
16	264	68
23	274	57
30	269	56
October 7	236	66
14	261	73
21	248	60
28	214	34
November 4	242	29
11	215	29
18	200	18
25	226	7
December 2	221	10
9	198	19
16	212	5

Buried in the 97 Pariſhes within the Walls 2696
Whereof of the Plague 190
Buried in the 16 Pariſhes without the Walls 4813
Whereof of the Pl. 603
Buried in the 9 Out-Pariſhes in *Middleſex* and *Surrey*, and at the *Peſt-houſe* 3045
Whereof of the Pl. 524
Buried in *Weſtminſter* 566
Whereof of the Pl. 31
The Total of all the Burials this time 10545
Whereof of the Pl. 1317

Burd of all Diſeaſes in he Year 1636.

	Total	*Pla.*
Apr 7	119	2
14	205	4
This wk these Pariſhes were added: S. Mag. Weſtminſter, Lambeth Pariſh, S. Mary Newington, Redriff Pariſh, S. Mary Iſlington, Stepny and Hackney pariſhes.		
Apl 21	285	14
28	259	17
Ma 5	251	10
12	308	55
19	299	35
26	330	62
Jun 2	339	77
9	345	87
16	381	103
23	304	79
30	352	104
Jul 7	215	81
14	372	104
21	365	120
28	423	151
Auguſt 4	491	206
11	538	285
18	638	321
25	787	429
September 1	1011	638
8	1069	650
15	1306	865
22	1229	775
29	1403	928
October 6	1405	921
13	1302	792
20	1002	555
27	900	458
November 3	1300	838
10	1104	715
17	950	573
24	857	476
December 1	614	321
8	459	167
15	385	85

The Total of the Burials this year is 23359
Whereof of the Pl. 10400

Buried of all Diſeaſes in the Year 1664.

	Total	*Plag.*
December 27	291	
January 3	349	
10	394	
17	415	
24	474	
31	409	
February 7	393	
14	461	1
21	393	
28	396	
March 7	441	
14	433	
21	365	
28	353	
April 4	344	
11	382	
18	344	
25	390	2
May 2	388	
9	347	9
16	353	3
23	385	14
30	399	17
June 6	405	43
13	558	112
20	611	168
27	684	267
July 4	1006	470
11	1268	727
18	1761	1089
25	2785	1843
Auguſt 1	3014	2010
8	4030	2817
15	5319	3880
22	5568	4237
29	7496	6102
September 5	8252	6988
12	7690	6544
19	8297	7165
26	6460	5533
October 3	5720	4929
10	5068	4327
17	3219	2665
24	1806	1421
31	1388	1031
November 7	1787	1414
14	1359	1050
21	905	652
28	544	333
December 5	428	210
12	442	243
19	525	281

The Total of the Burials this year is 97306
Whereof of the Pl. 68596

Place this Table at *page* 14.

Figure 2-2: Mortality Table. Source: (Graunt, 1676)

with the foresight to collect data even when the full potential of the data is not yet known, and a curious mind with access to said data willing to inspect it and convert it into usable information. In this case, equipped with a historic dataset (the mortality bills), tools (pen and paper) and his ingenuity, Graunt in the 17th century, was able to extract insights hidden in data that saved people's lives.

Even though the data that John Graunt analyzed spanned 50 years (Morabia, nd), he was able to manually summarize and analyze the data because each observation had a small number of features. The process of manually aggregating, summarizing, and analyzing collected data would remain a manual process until 1880.

2.2.2 Automated Data Processing

The next leap in data processing would once again emerge from the study of demographics. This time in 1880 at the hand of Herman Hollerith (1860-1929). Hollerith, while working at the U.S. Census Bureau, discovered that it would take 8 years to manually tabulate the 1880 census (Isaacson, 2014). It was also projected that the 1890 census would not be completed before the 1900 census began. Since the U.S. constitution mandates a census every year, this would be a big problem (Anonymous, 2017b). Hollerith resolved to automate the process of tabulating the census. Historians debate whether he took inspiration from the Jacquard loom – a power loom that weaves pattern in fabric dictated by punch cards – or from the manner in which train conductors punch holes in customer tickets (Randell, 1982). Regardless of the true source of inspiration, Hollerith went on to design an electrical system that leverages punch cards to automatically tabulate the census. Thanks to his invention, the data for the 1890 census was tabulated in 1 year (Isaacson, 2014).

Prior to Hollerith's invention, collected data, like census information, was processed manually by an army of analysts. This made data processing economically expensive, time intensive and error prone. The reliance on manual processes also meant that as the dataset grows, the resources needed to process the data in adequate times would grow beyond economical feasibility. But with the introduction of the punch card, operators were able to process data 10 times faster than before (Anonymous, 2017b).

Hollerith's devices proved useful beyond census tabulation and were later used in business and inventory tracking. This marked the beginning of automated data processing and the company he founded would later become – after a series of mergers and accessions – International Business Machinery (IBM). With the invention of the tabulating machine, humans now had the ability to automate the data analysis of large amounts data (relative to that period). This invention would help business calculate large sums of data from 1890 to 1950, when commercial general purpose computers took over.

2.2.3 Emergence of Business Intelligence Systems

The proliferation of automated data processing devices presented a new opportunity to Hans Peter Luhn (1896-1964), an IBM researcher and manager of of the information retrieval division. In 1958, Luhn published a paper titled "A Business Intelligence System" that outlined the design of a system capable of (1) ingesting data from documents relevant to a domain, (2) performing statistical analysis on the content of the documents to extract salient content, and (3) pushing summarized data to interested parties to assist in their day to day decision making (Luhn, 1958). Luhn envisioned such a system deployed in government, law, commerce, industry, and other areas, to "apprehend the interrelationships of presented facts in such a way as to guide action towards a desired goal". Luhn's paper laid the foundation for Business Intelligence (BI) systems to come.

Presently, BI systems are employed in major organizations to guide decision makers. These systems have access to large amounts of raw data that relate to the organization's day to day. Data can be collected internally from across the organization (For Example: daily sales by branch, number of knee surgeries performed, total number of products shipped, etc.) or externally (For Example: industry benchmarks, competitor data, stock market, etc.). The data collected can be aggregated, combined with other facts and summarized into reports or active dashboards for consumption. Consumers of the information can drill up and down the data, while varying the dimensions to focus on the factors that affect the decision under consideration.

Several technological advances contributed to the increases the ubiquity of BI systems. These advances included breakthroughs in storage devices (Goda and Kitsuregawa, 2012), the creation of relational databases models (Codd, 1983), and advances in data visualization.

In the decades that followed Luhn's paper, BI became an integral tool in the arsenal of any organization. Without the power that it provides to decision makers, organizations would find themselves losing to competitors that are more adapt at harnessing the power of their data.

2.2.4 Big Data

At the turn of the 21st century, researchers lamented the lack of tools to process large datasets. Computers were able to generate large amounts of data, but it eluded analysis because it simply did not fit into the memory of even the highest end computers (Bryson et al., 1999). The solution to this problem would emerge from papers published by Google in 2003 and 2004.

As a search engine, Google needed to perform three tasks efficiently: (1) Crawl the entire web, (2) store the generated index, and (3) serve search requests from the index as quickly as possible (Mellor, 2010). Performing these tasks over 1 billion webpages - the number of webpages in 2000 (Alpert and Hajaj, 2008) - was resource intensive. Typically companies would have relied on pre-built rack servers, but Google discovered that keeping their search service free would be challenging if they rely on costly servers (Shankland, 2009). In addition, these servers would also becomes points of failure in their infrastructure. Google realized that the jobs it will execute can be parallelized, therefore, instead of buying expensive powerful machines, they can split the job over multiple smaller inexpensive commodity machines. In the end they opted to invest in a large number of commodity off-the-shelf x86 machines that are 5 to 10 times cheaper than pre-built servers (Mellor, 2010). The servers Google purchased early on had 160GB disk space and 4-8GB of RAM (Dean and Ghemawat, 2008). By April 2003, Google was powered by more than 15,000 of these comodity servers (Mellor, 2010).

Since no other company at Google's scale had used such an infrastructure, it had to invent the necessary tools. In 2003 the public learned how Google was leveraging this infrastructure when a paper titled "Google File System" (Ghemawat et al., 2003) was published by Google researchers. In the paper, Google described a new type of file system called Google File System (GFS). According to the paper, GFS is a "scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients" (Ghemawat et al., 2003). GFS presents the entire set of commodity servers as one large file system by abstracting the underlying infrastructure. It is built with the assumption that the underlying hardware will fail, thus failover and fault-tolerance are core tenants of its design. GFS enabled Google engineers to perform large scale data processing over large clusters of servers. The largest cluster at the time of publication was a 1,000 server cluster that provided 300TB of disk space (Ghemawat et al., 2003).

GFS shed the light on how storage was handled on a parallelized infrastructure but did not detail how data processing and analysis was performed on the stored data. The answer to that would come in another Google paper published in 2004 titled "Mapreduce: simplified data processing on large clusters" (Dean and Ghemawat, 2004). In that paper, MapReduce, a programming model for parallel processing of large datasets was unveiled. Prior to MapReduce, data analysts needed to develop special purpose computation for every analysis they wished to perform (Dean and Ghemawat, 2008). MapReduce instead suggests breaking down every data analysis job into a sequence of *map* and *reduce* jobs. A *map* job takes data as input and produces a key-value pair that represents the output of executing the function on the data. A *reduce* job on the other hand takes the output of a *map* job and summarizes the data based on logic specified by the data analyst. Under this model, data analysts need to provide the input data, dictate to MapReduce the sequence of *map*/*reduce* functions, and the code that implements each intermediate *map*/*reduce* function. MapReduce can then take this input and determine how to parallelize the execution over the distributed cluster. Since the underlying infrastructure uses commodity hardware, a data analysts that wishes to speed up his data processing, or expand the size of his dataset, only has to requisition new cheap hardware to extend his cluster. The MapReduce algorithms will automatically distribute his data and jobs over the expanded capacity. Thanks to this platform, Google was able to run a data analysis job over 403,152TB of data in 6.5 minutes using 394 commodity servers (Dean and Ghemawat, 2008).

Even though Google released the designs behind the tools it used in-house, it never released the tools themselves. But with both papers now public, the open source community had enough information to translate the papers into tools that can be used by the community. Their efforts culminated in the creation of Hadoop. which quickly enabled the same data analysis capabilities as Google. Practitioners could now leverage Hadoop to perform data analysis over Petabytes of information. It became so popular that by 2012, 25 of the fortune 50 companies were running Hadoop clusters (Anonymous, 2012).

2.2.5 The Cloud

As we saw in the previous section, the tools needed to perform analysis of large datasets was now available for anyone to use for free. But to be able to execute these tools on large datasets, data analysts needed access to compute resources where these tools can be deployed and executed. Unlike Google, typical companies do not have large infrastructures waiting to execute data analysis jobs. To perform any serious data analysis, companies would need to invest in their infrastructure. Nevertheless, even large corporations with dedicated datacenters require a lead time to fulfill any internal demand for computing resources. Moreover, when the extra compute resource is added to the datacenter, this capital expenditure can sit idle up to 80% of the time (Gillis, 2015). The solution to these problems would emerge from Amazon in 2006.

In 2003, Benjamin Black, at the behest of his manager Chris Pinkham, was working on optimizing the Amazon infrastructure. The goal was to abstract the infrastructure as much as possible to simplify usage by the Amazon team. After some progress, they decided to pitch the idea of exposing the infrastructure externally as a service to Jeff Bezos. Bezos liked the idea and tasked Chris with developing what would become Amazon's Elastic Compute Cloud (EC2) (Black, 2009). The service was released to to the public in 2006.

Amazon effectively opened up its highly optimized computing infrastructure for anyone to use. It exposed storage and computing resources on a metered basis. Anyone, from individuals to Fortune 500 companies, could register an account and start using the service in minutes. Users of the service have access to 76 compute instance types with varying compute capacity, memory and network performance (Anonymous, 2017a). Users can spin up large compute clusters in minutes, run any job they wish to execute, and then turn off the cluster – paying only for the time used. Following Amazon's success, Microsoft, Google, and Oracle would follow suite and each would establish their own competing public cloud service.

Thanks to Amazon, and the companies that followed it, anyone building a data pipeline now has access to unlimited resources, both from a storage and computational standpoint.

2.2.6 The Spring of AI

The next key evolution is the creation of the field of Artificial Intelligence (AI). This field emerged from a conference at Dartmouth organized by John McCarthy (1927-2011), Marvin Minsky (1927-2016), Claude Shannon (1916-2001) and Nathaniel Rochester (1919-2001) in the summer of 1956. The intent of the conference was to discuss the possibility of creating machines that can achieve human-level problem solving by unifying work from various fields (McCarthy et al., 2006). The conference was attended by leading researchers from the fields of mathematics, electrical engineering, neurophysiology and cognitive psychology, who would go on to become the leaders of this field.

Ever since that conference in 1956, AI practitioners have been attempting to build systems that can match – and surpass – human's ability at solving tasks that require cognitive capabilities. Effectively, the field has been attempting to imbue machines with the following capabilities:

Two approaches to building AI systems emerged in the 1950s and 1960s: symbolic vs subsymbolic.

Symbolic systems used a combination of algorithms, domain knowledge representations and heuristics to attempt to deliver the above capabilities. Developers of symbolic systems had to create symbolic knowledge representations of the domain where the AI will be deployed. This would entail codifying the objects, valid states, properties, interrelationships, and rules of the domain in a format that the AI system could traverse. When presented with an input, the system would leverage the algorithms to traverse the knowledge representation to accomplish the task at hand (Nilsson, 2010).

Subsymbolic approaches to AI on the other hand combined Neural Networks, probability and statistics to mimic human intelligence. Neural Networks had been

Capability	Description
Text Recognition	The ability to understand the content, context and ontology of textual information (Jurafsky and Martin, 2009)
Text Generation	The ability to generate meaningful text that is understandable by humans (Jurafsky and Martin, 2009)
Object Recognition	The ability to recognize and extract contextual information from objects in a scene (Szeliski, 2011)
Object Generation	The ability to generate objects that fit a target context in a scene (Szeliski, 2011)
Speech Recognition	The ability to transcribe and extract information from spoken language (Jurafsky and Martin, 2009)
Speech Synthesis	The ability to synthesize meaningful spoken language that is understandable by humans (Jurafsky and Martin, 2009)
Planning	The ability to formulate a sequence of steps that accomplish a desired end-state (Lee et al., 2015)
Inference	The ability to suggest and/or rank actions based on provided data (Gramatica and Pickering, 2017)
Learning	The ability to observe or perform an action followed by internalizing the results to assist in future actions (Higgins et al., 2017)
Pattern Detection	The ability to surface patterns, correlations, and trends in data (Bishop, 2006)

Table 2.1: AI Capability

initially proposed by Warren McCullough (1898-1969) and Walter Pitts (1923-1969) in a paper titled *"A Logical Calculus of Ideas Immanent in Nervous Activity"* (McCulloch and Pitts, 1943). In that paper, McCullough and Pitts proposed constructing an information processing system that mimics the neurons in the brain. Frank Rosenblatt (1928-1971) would push their concept further by inventing the perceptron in 1957 (Rosenblatt, 1957). Rosenblatt's perceptron was the first model ever created to learn how to classify inputs from to a set of categories by being previously shown a sample from each category (Goodfellow et al., 2016). Neural networks proved itself well suited for human sensory tasks, such as recognizing objects and detecting patterns: as far back as 1969, neural nets were achieving 98% accuracy on handwritten character recognition (Nilsson, 2010) (Munson, 1968). Even though neural networks showed great promise, Marvin Minsky and Seymour Papert (1928-2016) would later publish a paper that shows a serious limitation in the perceptron that Rosenblatt created (Muller and Reinhardt, 1990).

The concept of Machine learning emerged during the early stages of AI as a methodology to teach machine to think by exposing them to large amounts of data (Nilsson, 2010). The term was first used by Arthur Samuel (1901-1990) in his 1959 paper (Samuel, 1959). It is important to note that Machine Learning spans both symbolic and sub-symbolic techniques.

After its inceptions, the field of AI would experience a series of ebbs and flows throughout the years:

After the second AI winter in 1993, research into AI fragmented across several disciplines such as computer science, mathematics, economics, operational research, and robotics. Advances in those areas made their way into everyday product and services that we use. For example, advances in vision where used to automatically check claims. Google used PageRank, an advance in recommendation systems, to build the best search engine. Credit card companies leverage AI techniques to combat

Period	Description
1950 - 1960	The field of AI is created. Researchers setup labs to focus on symbolic and sub-symbolic areas of AI. Neural networks are invented.
1960 - 1970	Major advances in vision, knowledge representation and natural language processing. Minsky and Papert expose a limitation in neural networks.
1970 - 1980	First AI Winter: limitations in both computing power and symbolic techniques meant a lot of promises made by AI researchers failed to materialize. Most of the advances developed proved successful only on toy problems in the lab and struggled in real-world settings. A new architecture of Neural Network was designed to tackle limitations, but interest in Neural Networks stagnate nonetheless.
1980 - 1990	Interest in expert systems revive interest in the field. This is followed by a second AI winter when to total cost of maintaining expert system becomes clear. Renewed interest in neural networks after a new learning algorithm was developed.
1990 - 2000	Increases in computing power overcomes some of the limitations that triggered the first AI winter. Research into AI techniques is now fragmented into various fields due to the second AI winter. DeepBlue beats Garry Kasparov. Emergence of self driving cars from DARPA challenge.
2000 - Present	AI Spring: Availability of large amounts of data coupled with increase in computing speed trigger renewed interest in the field of AI and Machine Learning

Table 2.2: History of AI

fraud on a daily basis. In addition, the reduction in cost of storage meant we now had large amounts of data to train sub-symbolic systems. This, coupled with the decrease in cost of computing and GPUs (Graphical Processing Units) to speed up model learning caused a resurgence in machine learning techniques. This phase is being dubbed the AI Spring.

2.3 Conclusion

In this chapter, we went over the importance of harnessing the large amounts of data that is being collected. In addition, we went over the definition of intelligent platform and detailed the technical advances that enable building these platforms. In the next chapter, we will take a deep dive into the technical components of intelligent platforms to understand how the concept discussed in this chapter manifest in the architecture of the platforms.

Chapter 3

Survey of Intelligent Platform Reference Architectures

3.1 Introduction

In this chapter we survey intelligent platform reference architectures from Microsoft, IBM, NTT Data and NIST. We intend to analyze each architecture to identify the ideal architecture to be used in developing our cost models. We found that none of the surveyed architectures where enough on their own to be adopted as our reference architecture, either because the components where not defined at a fine enough level of granularity, where coupled to vendor specific tools, or had a system boundary that would not be helpful in building the cost models. We were able to identify critical components that we will use in the subsequent chapter to develop our reference architecture.

It is important to note that the above mentioned organizations – except for the NIST – published these reference architecture as an effort to promote their products. This fact exposes the core weakness behind these reference architecture, as they will always be biased towards the vendor's tools and preferences.

For every architecture listed here, we will provide a brief summary of the role of each component and then detail what we believe the it did well and where it fell short. It is important to note that the analysis is seen from the viewpoint of developing a cost model. Our analysis would be different if we were analyzing these architecture based on a different criteria.

3.2 Architecture Comparison

The architectures in this section will be assessed based on the following criteria :

- Vendor Agnostic: This criterion determines how coupled the architecture is to a specific vendor tool or platform.
- Data Source Ingestion: This criteria assesses the diversity of data sources that the platform is capable of ingesting from.
- System Boundary: This criterion determines if the architecture contains components that are not related to intelligent platforms.
- Component Granularity: This criterion is used to assess the level of granularity that each components is defined at.
- Component Completeness: This criteria is used to determine if a core capability is missing from the platform.
- Integration Support: This criterion determines how well the architecture enables other services and applications.

We will summarize the assessment of the architectures by scoring them against each criteria on a scale from 1 to 5. A score of 5 indicates that architecture perfectly satisfies the criterion. The full comparison summary can be see in figure 3.1.

	NIST	Microsoft	IBM	NTT Data
Vendor Agnostic	5	3	5	4
Data Source Ingestion	2	3	5	3
System Boundary	2	2	2	4
Component Granularity	1	2	4	4
Component Completeness	2	3	4	4
Integration Support	2	2	3	5

Table 3.1: Architecture Comparison

3.3 NIST Big Data Reference Architecture

In 2015 the NIST's Big Data Public Working Group published the first version of their big data reference architecture (NIST Big Data Public Working Group, 2015c). The NIST developed the reference architecture by surveying 9 big data architectures provided by large organizations (NIST Big Data Public Working Group, 2015b) and 51 Big Data use cases collected from various industries (NIST Big Data Public Working Group, 2015a).

The published reference architecture outlined critical components that were common in the surveyed architectures. These components include:

- System Orchestrator: The system orchestrator is responsible for combining the various components and functions in the platform the deliver the needed business value. The orchestrator can be a human or a software component that coordinates the workflows.

- Data Provider: The data provider is any data source that provides new information to the platform. This can either be an external system, internal component to the platform or human operator. The data can arrive at different velocities and can be structure or unstructured based on the data source.

- Big Data Application Provider: This component is responsible for executing

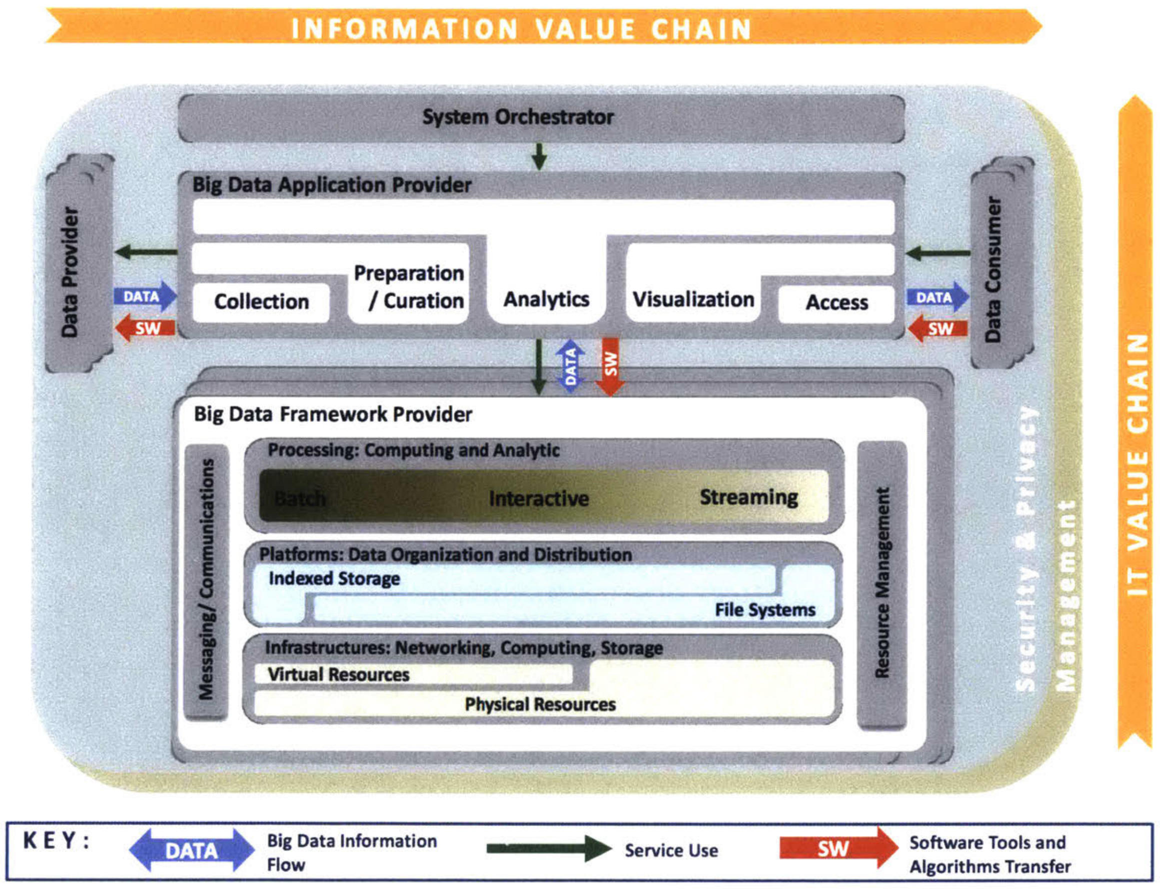


Figure 3-1: NIST Big Data Reference Architecture. Source: (NIST Big Data Public Working Group, 2015c)

the workflows dictated by the system orchestrator. This is done by collecting the data needed, preparing it for analysis by applying required transformations, analyzing it based on supplied criteria, visualizing it to communicate results to analysts and finally providing the result of the analysis to interested stakeholders.

- Big Data Framework Provider: The big data framework provides the shared infrastructure, data platform and processing frameworks that supports all the components of this reference architecture.
- Data Consumer: The data consumer is defined in the NIST reference architecture as either another system or a human end-user. This encompasses any entity that will consume the information produced by the platform and benefit from the analysis.

The NIST reference architecture suffers from 2 core weaknesses that stop us from adopting it as the basis of our cost mode. First the components of the platform, as they are decomposed by the report, are not granular enough. The architecture does not break down the components at a level that allows us to detail the cost model needed to deliver each component. Second, the architecture breaks down the components by roles to show that a component's action can be played by a software system or a human actor. This prevents us from drawing a system boundary around the software system.

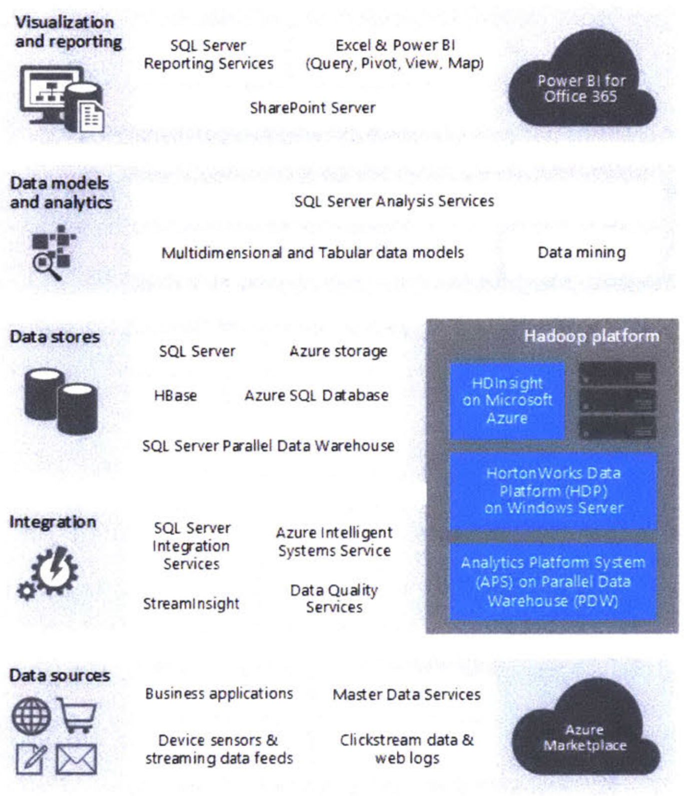
3.4 Microsoft's Big Data Reference Architecture

Microsoft's Big Data Reference architecture is broken down in 5 core components:

- Data Sources: This layer of the architecture shows the various data sources that a big data platform can integrate with.

- Integration: The integration layer is the interface between external data sources and the big data platform.
- Data Stores: The data stores contain the structured, semi-structued and unstructured data stored in the platform.
- Data Model and Analytics: This layer contains the data models that are used to resolve the queries made to this platform.
- Visualization and Reporting: The visualization and reporting layer encompasses all the tools that are used to show users the results of executing their queries.

The Microsoft reference architecture does a good job of showing the various types of data sources that a platform can ingest from. This architect also clearly shows the need for a data quality services sub-component to control the quality of data coming from the data sources. On the other hand, this architecture suffers from three core weaknesses that prevents it from being adopted as the basis for developing a cost model. The first weakness is due to including the data sources into the system boundary of this platform. This is a big flaw because it eliminates the interface from the architecture. Instead, this architecture should have explicitly added a data ingestion layer as the interface with the external data sources. We assume that this was done to show the the integration with Azure Marketplace to source datasets. The second weakness is the lack of an API component that allows the data models and analytics to be used by other services and applications. This architecture focuses more on generating reports and integrating into Microsoft's products (Example: Office, Power BI, SQL Server). It does not show how the data stores, the model and analytics layer can be shared by other services and applications. It is critical to design the system with these shared components in mind, as adding them later is too a complex of an undertaking. The third weakness is that the platform is too vendor specific;



Visualization and reporting
SQL Server Reporting Services
Excel & Power BI (Query, Pivot, View, Map)
SharePoint Server
Power BI for Office 365
Data models and analytics
SQL Server Analysis Services
Multidimensional and Tabular data models
Data mining
Data stores
SQL Server
Azure storage
HBase
Azure SQL Database
SQL Server Parallel Data Warehouse
Hadoop platform
HDInsight on Microsoft Azure
HortonWorks Data Platform (HDP) on Windows Server
Analytics Platform System (APS) on Parallel Data Warehouse (PDW)
Integration
SQL Server Integration Services
Azure Intelligent Systems Service
StreamInsight
Data Quality Services
Data sources
Business applications
Master Data Services
Device sensors & streaming data feeds
Clickstream data & web logs
Azure Marketplace

Figure 3-2: Microsoft's Big Data Reference Architecture. Source: (Microsoft, nd)

Microsoft built this architecture with the intent of mapping the components of the platform to it's offering. This disqualifies it for a general reference architecture since implementors will have different preferences for the software stack to use.

3.5 IBM's Big Data Reference Architecture

IBM's reference architecture breaks down the big data platform into 8 core components:

- Data Sources: This components lists the data sources that can be ingested by the a big data platform. It provides a detailed list of sample data sources for each sub-component listed in this layer.

- Data Massaging and Storage Layer: This layer is responsible for converting the data ingested from the data sources – arriving at various velocities – into a format that can be processed by the downstream layers. It is also responsible for storing the data in a raw and converted format for later retrieval by downstream layers.

- Analysis Layer: The analysis layer contains the business logic that converts the data into actionable insights. It also stores the models that run on the streaming and batch data received from the upstream layers.

- Consumption Layer: The consumption layer is responsible for exposing the result of executing the models and business logic on the data. It makes this information available for consumption by humans and other systems.

- Integration Layer: This layer allows all the above mentioned components to interact with each other. This is achieved using standardized protocols and API contracts.

- Big Data Governance: Big data governance permeates the entire platform. It is responsible for defining the policies that relate to data handling, retention and regulatory compliance.

- Quality of Service Layer (QOS): The QOS layer handles the security, quality of data, and service level agreements throughout the platform. It sets the policies related to privacy, security, data filters and masks.

- Systems Management: System management is responsible for handling the overall infrastructure, system monitoring and policy management.

IBM's reference architecture does a great job in several areas. First, it provides a comprehensive list of data sources that can be critical for a big data platform.

Not only does it provide a high-level view of these data sources, but it also details lists of data sources from each category. Second, the reference architecture explicitly mentions model management as a sub-component of data analysis. This is a critical part of data analysis and IBM's architecture is the only one that explicitly calls it out. Third, this architecture provides a comprehensive list of analytical techniques in the analysis engine. It explicitly mentions data analysis using complex event processing, and model execution. Fourth, IBM includes the 4 critical components that are needed to run any big data platform. These include : integration layer, big data governance, systems management and quality of service (QOS). Lastly, the architecture is vendor agnostic and the layers mentioned can be implemented using a wide range of available technologies.

Even thought IBM's architecture has a lot of positives, it is not suitable as the basis for a cost model. Similar to other architectures, IBM includes the data sources as part of the system boundary of the architecture. Our reference architecture will need to remove these data sources from the architecture, and instead focus on the interface between those systems and the components included within the boundary of the platform. The second problem is that the architecture does not decompose the data massaging components into detail sub-components. it glosses over the details of data masking, data conversion and how data arriving at various velocities are converted into a usable format. It also neglects to detail the types of data storage needed to support such a platform.

3.6 NTT Data's Big Data Reference Architecture

NTT Data breaks down its big data reference architecture into 8 core components:

- Information Gathering: This components is responsible for aggregating and validating raw data from various data sources. It exposes the data collected for

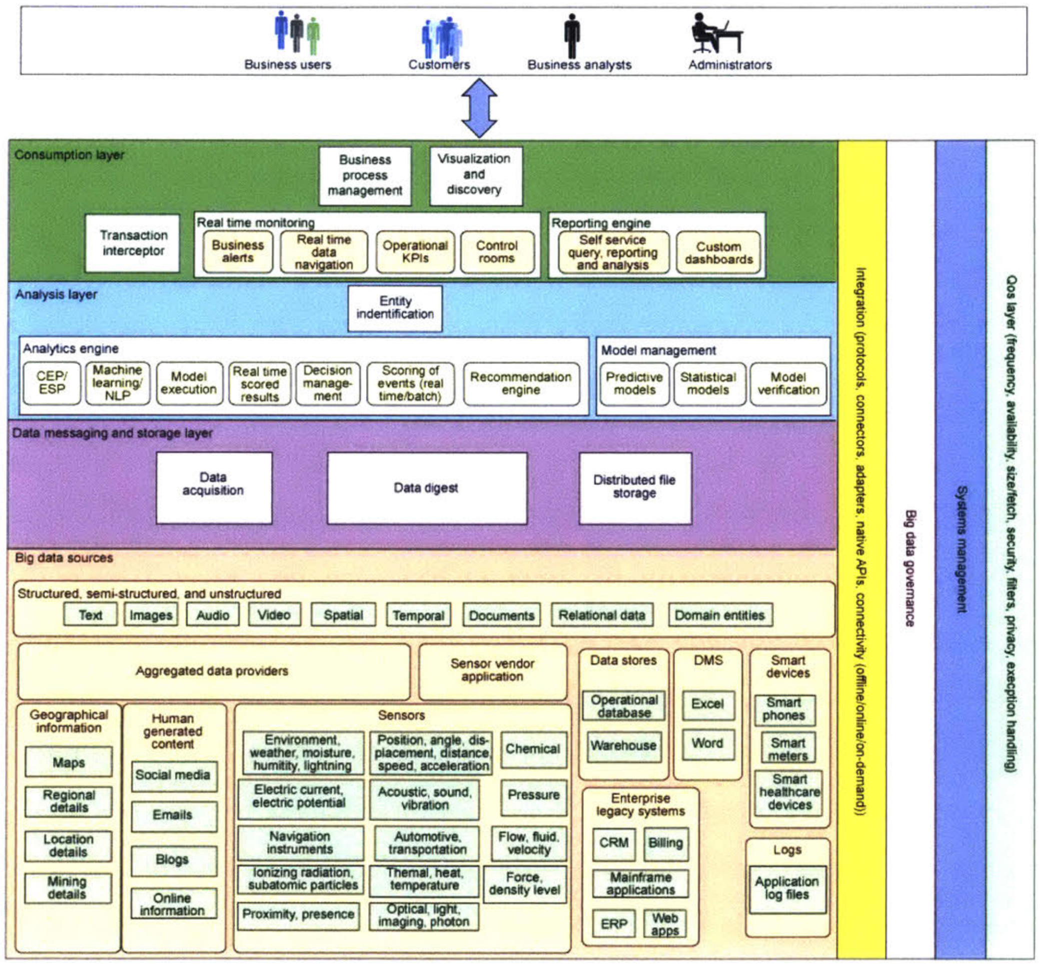


Figure 3-3: IBM's Big Data Reference Architecture. Source: (Mysore et al., 2013)

data processing and storage by downstream layers.

- Data Processing: Data processing is responsible for ingesting data from the information gathering layer, processing it, and transforming the data into a format that can be used for analysis.

- Information Store: The information store layer is used to persist the raw and processed data. The data can be stored in a structured, semi-structure and unstructured format.

- Data Analytics: Analysis of the data is performed using machine learning techniques, simulation and using various mining techniques.

- Analytics Methodology: This is a proprietary analytics methodology that is developed by NTT data.

- Decision Support and Utilization: This layer is responsible for exposing the processing data for use by external consumers.

- Governance: The governance component of this architecture is responsible for ensuring the security of the platform, maintaining data quality and managing the data lifecycle. It spans all the components of the architecture.

- Infrastructure: The infrastructure layer encompasses all the hardware and software components that are necessary to run the platform.

NTT Data's reference architecture provides a comprehensive list of components needed to deliver a reference architecture. It does this while providing a fine enough level of granularity that would enable an implementor to understand the necessary components. It also leaves the big data sources outside the system boundary. That being said, NTT Data does sugger from several issues that prevent it from being used

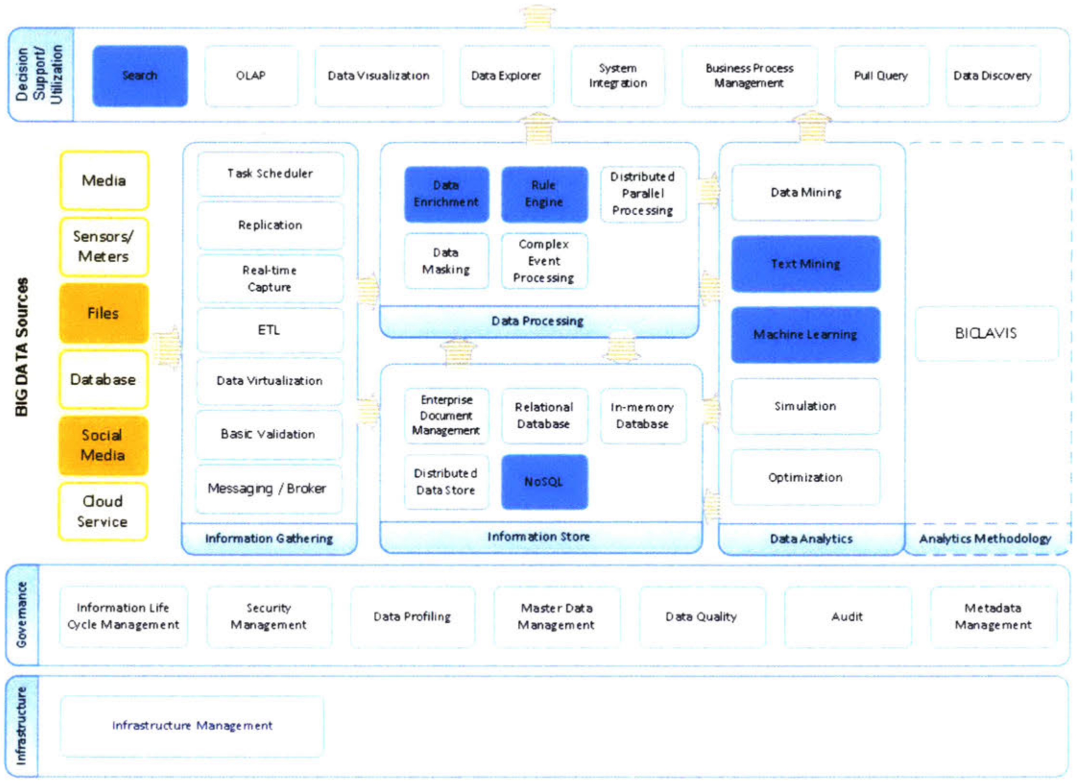


Figure 3-4: NTT Data's Big Data Reference Architecture. Source: (NTT Data Corporation, 2015)

as is. The first problem is that the sub-components detailed for the data analytics capability are not comprehensive: it does not list all data analysis sub-components that can be used to analyze the ingested data. It also includes a non-vendor specific implementation – BICLAVIS – without detailing non-vendor specific equivalents. Another problem is that the data analysis component mentions Machine Learning as a broad capability, but does not call out model execution and model management as a needed capability. Finally, the reference architecture does show interactions between the components, but does not explicitly define what each interaction is.

3.7 Conclusion

Even though we were not able to identify a reference architecture that we can use as is, thanks to Microsoft, IBM, NTT Data and NIST's reference architectures we can now extract the core components that an ideal intelligent platform should implement. In the next chapter, we will build on the knowledge gleaned from these architecture to present the reference architecture that we believe will help us develop the cost models.

Chapter 4

Intelligent Platform Reference Architecture

4.1 Introduction

In this chapter we will outline a reference architecture for intelligent platforms. We intend to use this holistic architecture throughout this thesis to define the core components needed to deliver an intelligent platform and subsequently determine the cost model of each component. To develop this list we extracted core components from the architectures reviewed in the previous chapter and added components that we found critical when delivering intelligent platforms in our professional experience.

4.1.1 Definition of Reference Architecture

Throughout this thesis, we will adopt Cloutier et al.'s definition of reference architectures:

> "Reference Architectures capture the essence of existing architectures, and the vision of future needs and evolution to provide guidance to assist in

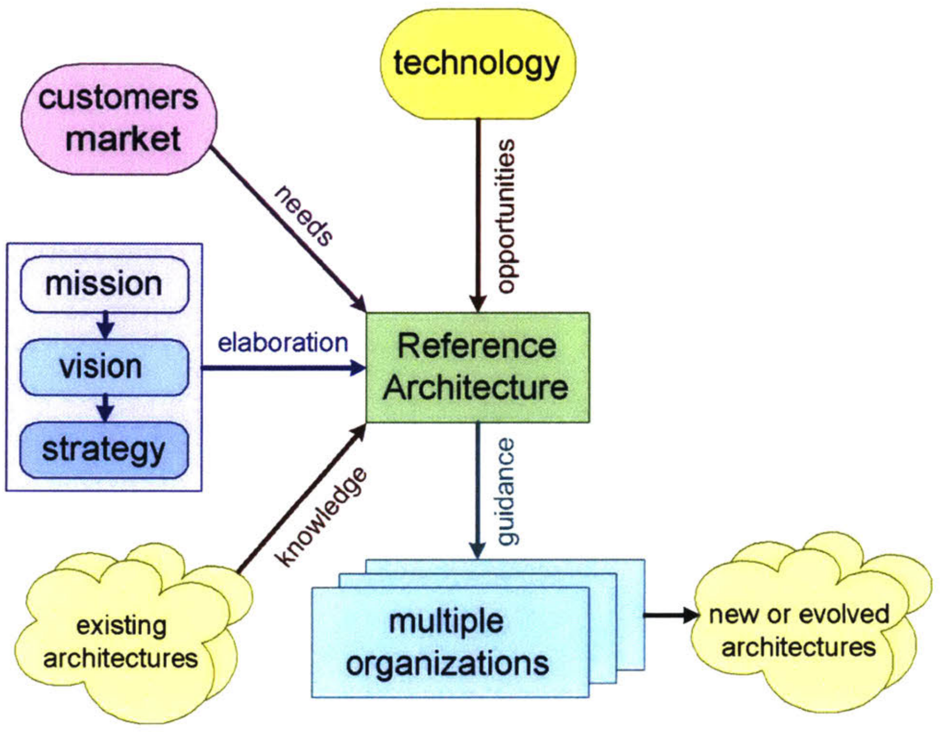


Figure 4-1: Framework for Reference Architectures. Source: (Cloutier et al., 2010)

developing new system architectures."

We do not expect the reference architecture detailed here to be implemented as is. Instead, we expect it guide the development of an organization specific architecture that incorporates all the business needs and vision of the implementing organization. Therefore, the reference architecture should be technology and vendor agnostic, giving organization the freedom to select the appropriate vendors and tools.

4.2 Reference Architecture

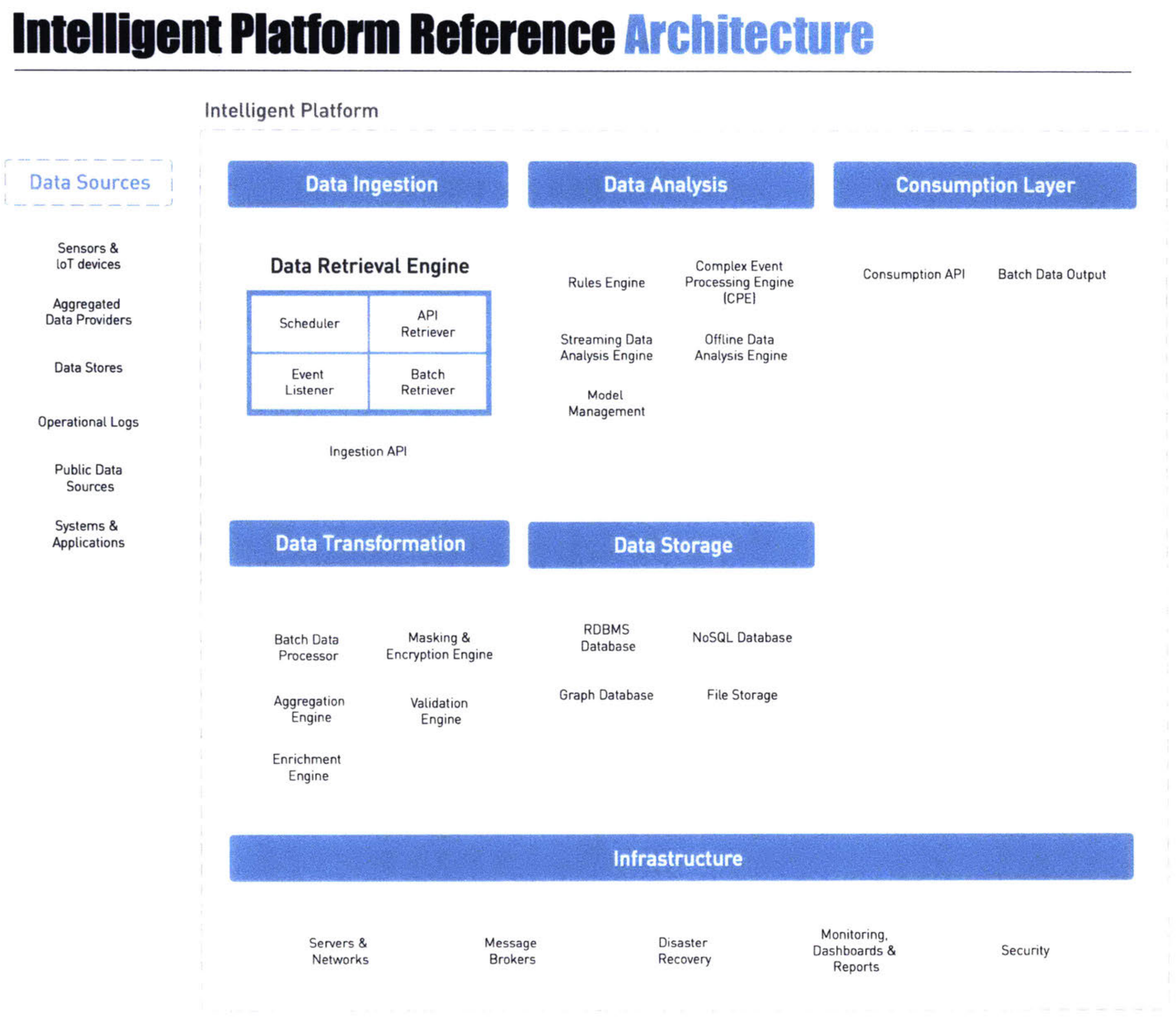


Figure 4-2: Reference Architecture for Intelligent Platforms

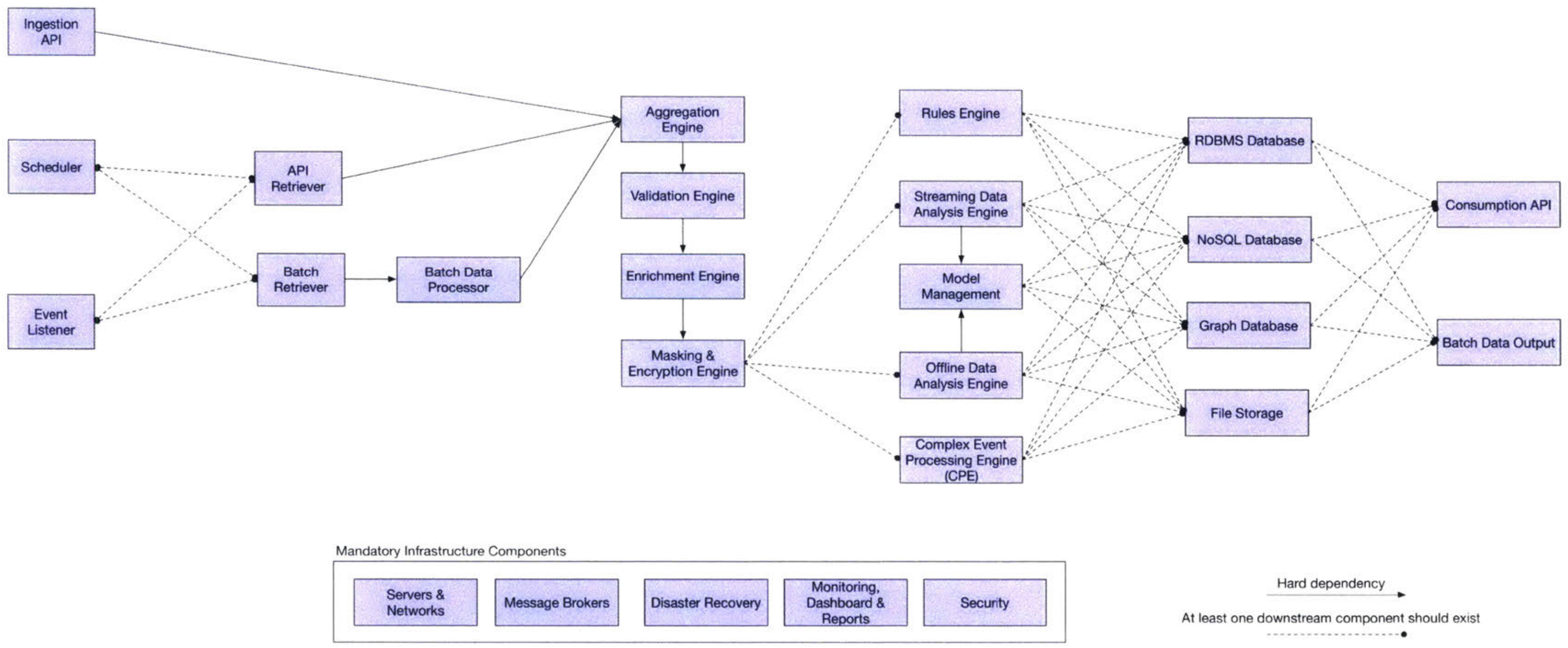


Figure 4-3: Reference Architecture Dependency Map

4.3 Data Ingestion

The first core layer in our architecture is the data ingestion layer. This layer sits at the boundary of the platform and acts as the interface for ingesting data from various sources. The sub-components in this layer can consume data in a push or pull paradigm; some of the information needed by the platform will be pushed into it by external data sources, whereas other data sources will require the platform to explicitly pull the data it needs. The latter will be triggered by an event – external or internal to the platform – or based on a predefined schedule.

4.3.1 Data Sources

The data sources are outside the boundary of our platform, but the interfaces with them are not. Therefore it is critical to list the types of data sources that this platform can interact with. In addition, the velocity of data sources that an organization wishes to consume from can heavily dictate the selected sub-components. For example, an organization wishing to consume data from high-velocity sources – such as IoT devices – will need an ingestion layer that allows it to ingest and propagate the data downstream to the analysis layer with sub-second latency. A platform that will be consuming slow moving data on the other hand, does not need such strict latency requirements. To understand the sub-components needed by an organization, we will categorize the data sources along the dimensions defined in table 4.1.

We believe a reference architecture should support ingesting data from the below listed sources.

Sensors & IoT devices

The the proliferation of IoT devices has created an opportunity for organizations to collect data from their products in real-time. An intelligent platform should be

Criteria	Description	Values
Velocity	This criteria qualifies the speed at which the data consumption should happen to extract the most utility from the data source (Rubinfeld and Gal, 2017)	**Low-Latency**: The data should be consumed in near real-time (sub-second) latency to extract maximum utility. **High-latency**: The data can be consumed without any real-time constraints.
Access	This criteria qualifies if the data is pushed by the data source into the platform, or if the platform itself has to pull the data from the source	**Push**: The external data source is responsible for pushing the data into the platform. **Pull**: The platform must pull the information it needs by reaching out to the data source in question.
Structure	This criteria determines if the data source holds structured or structured data (Rubinfeld and Gal, 2017)	**Structured**: The data held in this store has metadata associated with each element. **Unstructured**: The data in this store has unmarked text, video, or audio content. The data elements will need to be identified to perform further analysis.

Table 4.1: Data Source Dimensions

able to ingest data from these data sources and make it available for analysis as close to near real-time as possible. We define near real-time as sub-second latency from the moment the data is received by the platform to the time it is available for analysis by downstream layers. This data would be considered streaming real-time data. An example of this data source includes smart meters that can push the current electrical consumption at near real-time intervals, telemetry data from cars, and GPS information from shipping containers.

Data coming from these data sources is usually structured information as it is almost always pushed into the platform via its exposed APIs. It is is not advisable to have unstructured data coming from these data sources as the transformation steps needed to convert this to structured data for analysis can add to the latency.

Criteria	Value
Velocity	Low-Latency
Access	Push
Structure	Structured

Table 4.2: Sensors & IoT

Aggregated Data Providers

An organization might need to consume from entities that aggregate data for a certain domain. These entities could be governmental, non profit organizations, or third party data vendors. The data they offer can be instrumental in enriching an organization's existing information.

The data ingested from these sources are typically high latency data due to the nature of the work needed to aggregate, transform and expose the data. Organizations that wish to ingest data from these sources typically have to pull in the data into the platform either by calling APIs exposed by the external entity or by consuming batch files. By definition, this data is structured to enable consumption by external entities. Examples of such data sources include population health information, or geospatial information.

Criteria	Value
Velocity	High-Latency
Access	Pull
Structure	Structured

Table 4.3: Aggregated Data Providers

Data Stores

An organization might have a number of internal or external data stores that contains information of interest. These stores might be repositories of information for disparate business units, or repositories that can be accessed under data sharing agreements between the data store owner and consumer. This data is often pulled by the platform based on a set periodicity or in response to an event.

Data ingested from these sources could be either low or high latency based on the type of data in the stores, and the use cases. The data stores could be relational databases, NoSQL databases, or raw file stores. Relational data stores would contain structured data, whereas NoSQL databases and raw files could contain either structured or unstructured data based on the associated metadata. Examples of these data stores include relational databases used by an organization's finance department or a hospital's medical notes on patient visits.

Criteria	Value
Velocity	High-Latency & Low-latency
Access	Pull
Structure	Structured

Table 4.4: Data Stores

Operational Logs

A large organization typically has a large number of software application continuously generating operational logs. These logs capture user interactions with the software product, the errors encountered and any other information that the developers have decided to output. This data can help an organization understand how their customers are using their products. In addition, this can be instrumental in detecting problems or bugs in the deployed systems. For example, in the early data of Google, a developer inspected users' clickstreams after they performed search to track the relevance of the returned search results (Levy, 2011).

The data from these logs is often pushed into the platform in near real-time by a log aggregator (a software application that sits outside the boundary of the intelligent platform). All applications in an organization are typically programmed to forward their logs into the aggregator. It then forwards the logs to designated endpoints – including the intelligent platform.

Based on the amount of data transformation performed in the log aggregator, the data can be structure or unstructured. Adding too many costly transformations in the log aggregator can delay the data from reaching the analysis layer of the platform.

Example of this data can be the position of the user's cursor, click coordinates on a web page, screens visited on a mobile application, and intrusion detection logs from a company's mission critical servers.

Criteria	Value
Velocity	Low-latency
Access	Push
Structure	Unstructured

Table 4.5: Operational Logs

Public Data Sources

Public data sources present the broadest data source category in our list. It encompasses social media, blogs and public websites. Data extracted from these data sources can have a huge impact on the type of analysis that an organization can perform on. For example, an organization can run sentiment analysis on social media posts around the launch of a product or marketing campaign to track efficacy and engagement. They can also use this channel to detect customer complaints and golive issues early on. This allows them to be pro-active in addressing brand damaging problems early on.

The highest utility can be extracted from this data when the platform pulls it in real-time. Data sources could be accessible using public APIs (Example: Twitter API, Facebook API, Weather Channel API) or scrapped from the original data source to extract the needed information. If the data was pulled from APIs, then it is often structured, whereas scrapped data is unstructured and requires transformation before analysis. Some examples of these sources include, Twitter, Facebook, Yelp, Weather.com, Amazon and Reddit.

Criteria	Value
Velocity	Low-latency
Access	Pull
Structure	Structured

Table 4.6: Public Data Stores

Systems & Applications

An organization's internal systems and applications can be an excellent data source. They can include Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) systems, billing systems or any other custom built or acquired applications used by the organization.

The data from these platforms can be pushed or pulled based on the use cases. These products typically contain plugins that allow external application – like the intelligent platform – to pull data on demand. They can also be programmed to export data on a pre-set schedule and push it to an endpoint. Since organizations typically own these products, they can dictate the ideal data retrieval mechanism for the use cases in question. The data is often structured and ready for analysis by the platform.

Criteria	Value
Velocity	Low-latency
Access	Pull & Push
Structure	Structured

Table 4.7: Systems & Applications

4.3.2 Data Retrieval Engine

The data retrieval engine is the sub-component in the data ingestion layer responsible for pulling data from external sources. It fulfills all the use cases where the sources were marked with a pull access in section 4.3.1. The retrieval process can be triggered by a schedule provided to this sub-component or by a stimulus from an event originating inside or outside the platform.

To fulfill its role, this engine relies on the following critical sub-components:

Scheduler

The scheduler maintains a list of data sources and its respective data retrieval schedule. When it is time to pull a piece of information into the platform, it triggers the API retriever or the batch retriever depending on the type of retrieval mechanism exposed by the source in question.

Event Listener

The event listener idly waits for a predefined set of events to trigger. These events can originate internally from the platform, or from an external source. When a recognized event is received, it triggers – similar to the scheduler – the API retriever or the batch retriever depending on the type of retrieval mechanism exposed by the data source in question.

API Retriever

The Application Programming Interface (API) retriever is responsible for pulling information from an external data source that has an API exposed. It is responsible for maintaining the API contract information in order to request the needed information when triggered. The data is then pushed to the Data Transformation Layer (Section 4.4).

Batch Retriever

The Batch Retriever maintains a list of all sources that expose data batched in files. When triggered, it reaches into to the data source in question and retrieves the stored files. The Batch Retriever will then pass the files to the Batch Data Processor component (Section 4.4.1) for transformation.

4.3.3 Ingestion API

Data sources can push data into the platform using an Application Programming Interface (API). Exposed APIs accept structured data that adheres to its contract. It is responsible for receiving data from all sources listed in section 4.3.1 marked with a push access. When data is received from a source, this layer passes it to the Data Transformation Layer (Section 4.4) for further processing.

4.4 Data Transformation

The data used for analysis will arrive into the platform at various velocities and from a large number of sources with varying structures. The purpose of this layer is to normalize the data into a final structure that can be used for data analysis. Once the ingested data goes through all the transformations listed in this section, it will be ready for consumption by the Data Analysis layer (Section 4.5)

4.4.1 Batch Data Processor

The batch data processor layer is responsible of extracting the needed information from the batches of files retrieved by the Batch Retriever. This layer holds the expected structure of the input files and the set of transformations that enable it to convert the data into individual records. For this transformation, we will follow the architectural pattern set in the paper "Data Ingestion for The Connected World" (Meehan et al., 2017). In the paper, the authors recommend having one common data transformation pipeline for real-time data and bulk data. This architectural decision is critical as it has ramifications on the complexity of data transformations and the complexity of interaction between components. This aligns with our professional experience. We observed that this approach reduces complexity of downstream processing by assuming that all data going into transformation will be in the same pipeline. Therefore, this component will breakdown the batch files into individual records – as if they were received individually from a real-time data source – and push them into the data transformation pipeline for further downstream processing.

4.4.2 Aggregation Engine

Related pieces of information can come from various data sources at different speeds. This layer is responsible for making sure related data fragments are aggregated into a unified structure. This is crucial to ensure that the analysis layer has an accurate picture when the analysis is performed. This layer will inspect the pipeline as data is flowing then stitch the data elements together to form the full picture.

4.4.3 Enrichment Engine

The data coming into the platform will rarely hold all the identifying information. More than often, the platform will need to enrich the flowing data with further information. For example, a bank transaction coming from a main frame might hold the accounts involved in the transaction, but not the customer identifiers. This layer will be responsible for looking up the data needed and injecting it into the flowing data. This layer will retrieve the necessary data from local stores (Section 4.6).

4.4.4 Masking & Encryption Engine

All ingested data will contain a large variety of information. Some of this information might be regulated, or the organization might want to prevent security compromises from exposing it. For example, personally identifiable information in healthcare is protected under the Health Insurance Portability and Accountability Act (HIPAA) in the United States. All HIPAA data needs to be encrypted when stored. This layer is responsible for identifying the pieces of information that require masking or encryption and applies the equivalent transformation.

4.4.5 Validation Engine

Data elements flowing in the the platform will need to adhere to a set of business and syntactical rules. For example, a date field must not have a month value bigger of 12 and an email address cannot contain two '@' symbols. These constraints are crucial to ensure the analysis is accurate. Such errors could have a catastrophic impact on reliability of the platform. Since the data will be coming in from various data sources that the consuming organization cannot govern, the validation layer is the defense against having these errors impact the platform. This layer will have a list of fields and their equivalent constraints. Any field that does not adhere to the constraints will be rejected and an alert raised for further investigation.

4.5 Data Analysis

The data analysis is the core layer that is responsible for extracting the insight from the collected information. This is where the culmination of the research done in AI and machine learning manifests itself. This layer holds all the components that allows a firm to turn the data into actionable insights.

4.5.1 Rules Engine

The rules engine sub-component is a store of rules hand crafted by domain experts. It monitors data elements flowing through the pipeline and triggers events or alerts based on the coded rules. For example, the rules engine might have a rule that instruct it to release pressure on a valve if it starts seeing temperatures for a boiler above a certain critical level. Another example is a medical diagnosis system that contains the rules and symptoms for various diseases. Such a system would be able to suggest a prognosis based on the observed symptoms by consulting its knowledge base.

The cost of codifying the entire rules of a domain is too prohibitive, but these engines can be great for centralizing a small set of rules that need to be shared by various processes across an intelligent platform. This would streamline changing the business rules and propagating them across the data analysis jobs that utilize them.

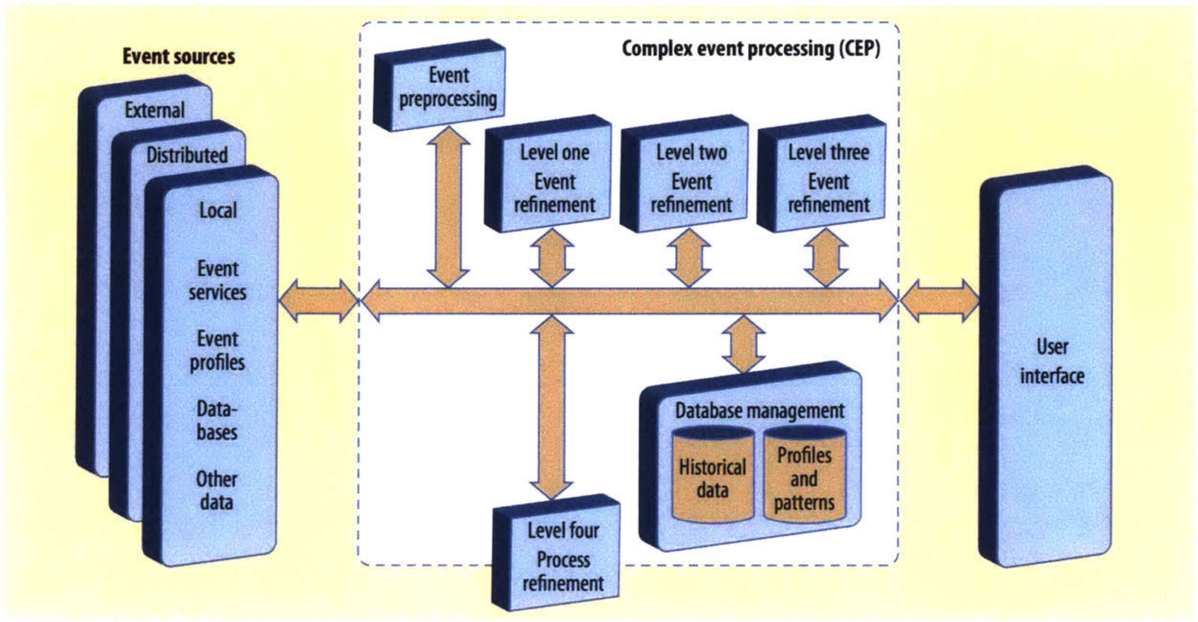


Figure 4-4: Complex Event Processing Engine. Source: (Leavitt, 2009)

4.5.2 Complex Event Processing Engine (CEP)

Complex event processing (CEP) engines extend typical rules engines by focusing on the relationship between real-time events flowing through the system. Such an engine is able to make inferences based on observing a sequence of events originating from disparate data sources (Leavitt, 2009). From example, A CEP engine can determine if a transaction is fraudulent by parsing – in real-time – all the financial events flowing throughout the system. Another heavy use of CEP is in cybersecurity, where CEP engines are used to correlate events occurring in an organization's infrastructure to detect attempts of cyber intrusion. CEP's are also powerful enough to send an alert when an expected event does not occur.

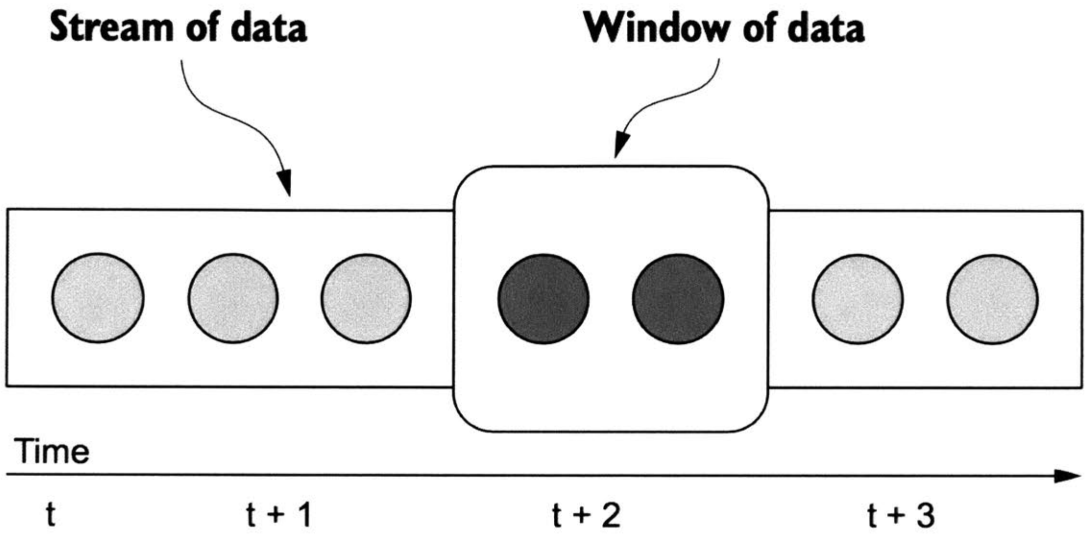


Figure 4-5: Streaming Data. Source: (Psaltis, 2017)

4.5.3 Streaming Data Analysis Engine

Streaming data analysis engines excel at running analytical tasks over data flowing into the platform in near real-time. These engines are typically distributed in-memory gird clusters, where an analytics job would be split over machines in the cluster. These engines excel at resolving continuous queries over the streaming data in addition to ad-hoc queries similar to RDBMS systems (Psaltis, 2017). This is achieved by retaining a number of data elements – called a window of data – in memory for quick access and query processing (Psaltis, 2017). The number of elements in a window of data is usually configurable but limited by the memory and processing capacity of the servers where the streaming engine is running.

Streaming engines are able to run any type of analysis on the data retrained in the window. This includes executing code from custom code, rules engines, CEP engines, and executing pre-trained models. That said, streaming engines also excel at answering summarization queries that span the full history of the data streamed.

Questions such as : how many times has event X occurred, or what is the event that occurred the most between two dates? This is achieved by leveraging probabilistic data structures like bloom filters (Bloom, 1970) and hyperloglog (Flajolet et al., 2007). For example, using these algorithms, streaming engines can determine the frequency of occurrence of one billion distinct items with an accuracy of 2% using only 1.5k of memory (Psaltis, 2017).

4.5.4 Offline Analysis Engine

Offline analysis engines are used to run analytical jobs that require an amount of data that makes it prohibitive to process it in near real-time. These engines usually have the same exact capabilities as the streaming data analytics engines – especially since a number streaming analytics engine support both offline and real-time analytics.

Similar to streaming data analytics, these engines can execute custom code, rules engines, CEP engines and pre-trained models over the data. They are typically executed periodically, for example executing fraud analysis over petabytes of data every night, or based on a trigger from a process in the platform. The result of the analysis is often stored back into the local data stores. The result of the analysis can also trigger events or alerts in the platform.

4.5.5 Model Management

One of the core capabilities needed in the platform is the ability to execute models on the ingested data. The term model is a highly overloaded and we were not able to find a single definition that accurately captures all the types of models that an intelligent platform should support. Therefore, for this platform, we will assume models can capture the following information :

- Summary statistics representing the probability distribution and their associated hyper-parameters

- Regression parameters
- Set of rules and branch values that define decision trees
- The network structure, weight values and classes that summarize a neural network
- Natural language processing domain datasets and their associated algorithms
- Clustering parameters
- Parameters representing recommender systems

A model management system should be able to store all the parameters associated with these models. It must expose the parameters for retrieval by the offline and streaming analytics engines. It should also be easy to update the parameters, and be able to run test to compare version of models against each other based on a supplied accuracy measure.

4.6 Data Storage

The data flowing into the system will need to be persisted for later access or offline analysis. The result of the analysis will also need to be captured for dissemination. Since we expect to ingest large amounts of data, these data storage layer should be able to hold large amounts of data – from terabytes to petabytes of information based on the organization. The data storage layer should be able to store data of varying types: text, binary, images, audio or video.

The data stored by the platform will become essential to the data scientists in their effort to optimize their models and tackle new business problems. Therefore

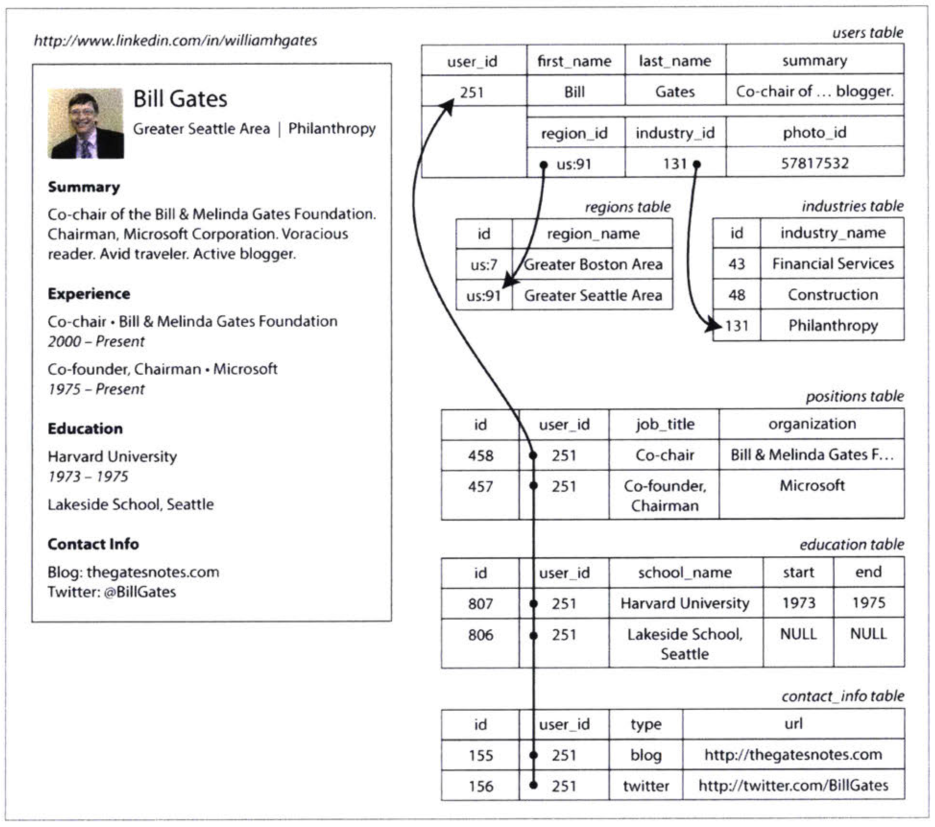
– in addition to storing this data in production – data scientists should be able to easily access the information they need to run experiments. This is further complicated when data stored is sensitive and cannot be openly shared internally in the organization. That said, the architecture of these platforms should make it easy for data scientist to run any experiments with little friction to make sure that the full potential of the data is unlocked.

Non-media data should be stored in relational database management systems (RDBMS), NoSQL or graph databases based on the data's size and use cases. Databases are inefficient for storing binary or media files. This type of information must be stored in a file system. A platform might commit to a single type of these databases or use a mix based on the needs and tradeoffs. None of these database technologies are dominant as each one offers varying tradeoffs as we will see in the following sections.

4.6.1 Relational Database Management System

In a relational database management system (RDBMS), the data is highly structured, has minimal data redundancy (normalized) with consistent relationship (imposed referential integrity). This is a strength for databases as the data models can mimic business models and impose constraints when inserting new data. This process of imposing the data integrity during write is called "schema on write". An example of a schema and sample data can be found in figure 4-6.

As an organizational's data needs start to expand, RDBMS system become difficult to manage (Bazar and Iosif, 2014). Changing the data structure (schema) becomes problematic as modifying the model would risk breaking the existing referential integrities and migrations between versions of the schema become a costly process. This causes organizations to avoid changing their schemas too much, thus reducing their agility in responding to business changes.



http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.
	region_id	industry_id	photo_id
	us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Figure 4-6: RDBMS example. Source: (Kleppmann, 2017)

```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University",       "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog":    "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```

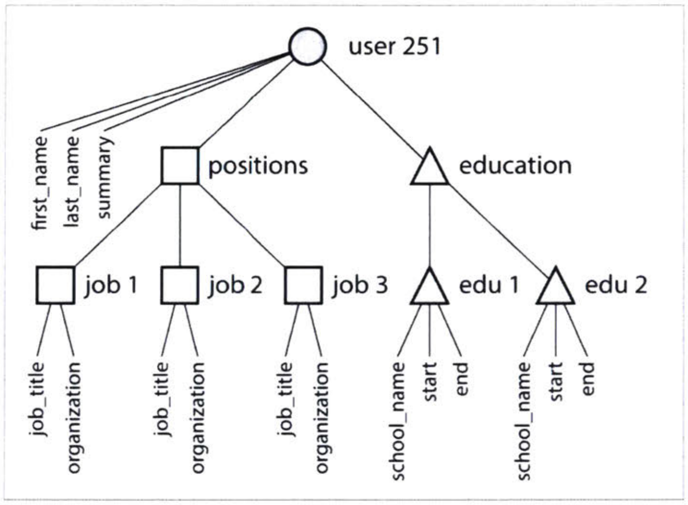


Figure 4-7: NoSQL Document example. Source: (Kleppmann, 2017)

4.6.2 NoSQL Databases

In contrast to RDBMS systems, NoSQL databases eschew schemas. Instead, they act as simple document or key/value data storage systems. Without performing any checks on the relationship between the data elements when the data is inserted. They assume that the relationship between the data elements is known by all data requestors. This paradigm is known as "schema on read". Figure 4-7 shows how the same data from Figure 4-6 can be captured in a NoSQL document store.

The lack of schema makes it easy to ingest large amounts of data without the need

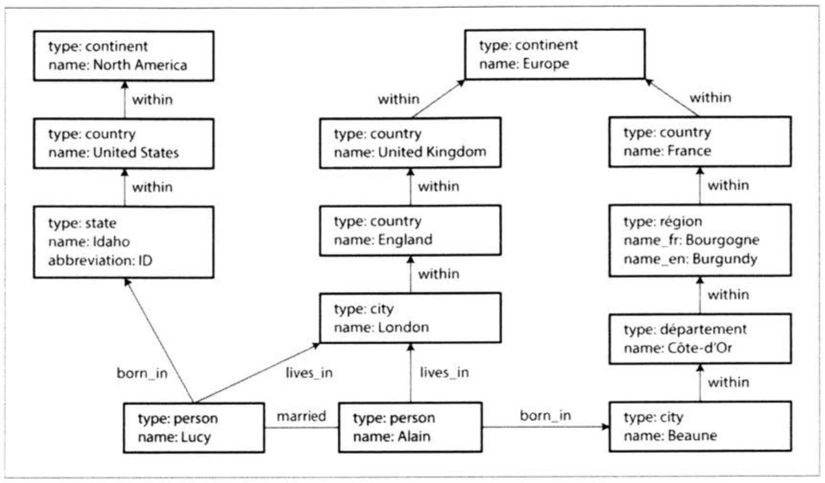


Figure 4-8: Graph Relationship example. Source: (Kleppmann, 2017)

to craft a schema that represents the relationship between the elements (Bazar and Iosif, 2014). The major problem with NoSQL databases is the complexity associated with ensuring data consistency across all data records, Since most NoSQL databases do not ensure transactional consistency.

4.6.3 Graph Databases

Graph database attempt to capture the relationship between data elements to make it easy for traversing the data stored based on the relationship. Since the data is captured as a connected graph, determining relationships between entities is done by traversing the edges that connect them. This makes it easy to answer queries that are focused on the connection between data elements. Therefore they are excellent at modeling domains where many-to-many relationships dominate (Kleppmann, 2017).

The weakness of graph databases lies in the difficulty in performing queries that require aggregating data (Kleppmann, 2017). Due to the data being captured in a graph format, queries that requires merging information about multiple unrelated relationships are complex to implement.

4.7 Consumption Layer

The consumption layer is responsible for (1) exposing the data collected, (2) making the data analysis accessible to external consumers, (3) and allowing external services to trigger data analysis jobs.

4.7.1 Consumption API

The consumption API is a service contract that allows external applications to interact with the platform. This the core component that enables internal and external application developers to build products using this platform. This API exposes the full capabilities that the platform offers.

4.7.2 Batch Data Output

Just like external data sources can push data into the intelligent platform, the platform itself should be able to export bulk data to external data sources. Similar to the ingestion batch components, this too can be triggered using an event or based off of a schedule.

4.8 Infrastructure

To run all the software needed to deliver the components mentioned so far, we will need an underlying infrastructure of hardware and software, and the necessary governance policies.

4.8.1 Servers & Networks

An organization will need access to servers that are able to execute the software necessary to power this platform. Due to advances in cloud computing, organizations now have almost unlimited on-demand infrastructures that can be spun up in minutes. By leveraging a cloud infrastructure, an organization can dynamically respond to spikes in demand by automatically provisioning the hardware needed. The resources can then be reclaimed when the demand subsides to reduce the operational cost of the platform.

4.8.2 Message Brokers

Message brokers are essential in an intelligent platform due to the distributed nature of the components that will be running. Message brokers act as a conduit that ties all the components together. Components subscribe to message topics where they receive the data needed to operate. When a message is pushed to that specific topic, all subscribed services are alerted. They subsequently retrieve the pushed message, perform their operation, and then pass the result of their operation into a subsequent message topic to alert downstream components.

4.8.3 Disaster Recovery

The platform, ingested data and analysis performed by this platform will become a core asset for the organization. Therefore special process should be put in place to make sure the platform is resilient against external events that can adversely affect it.

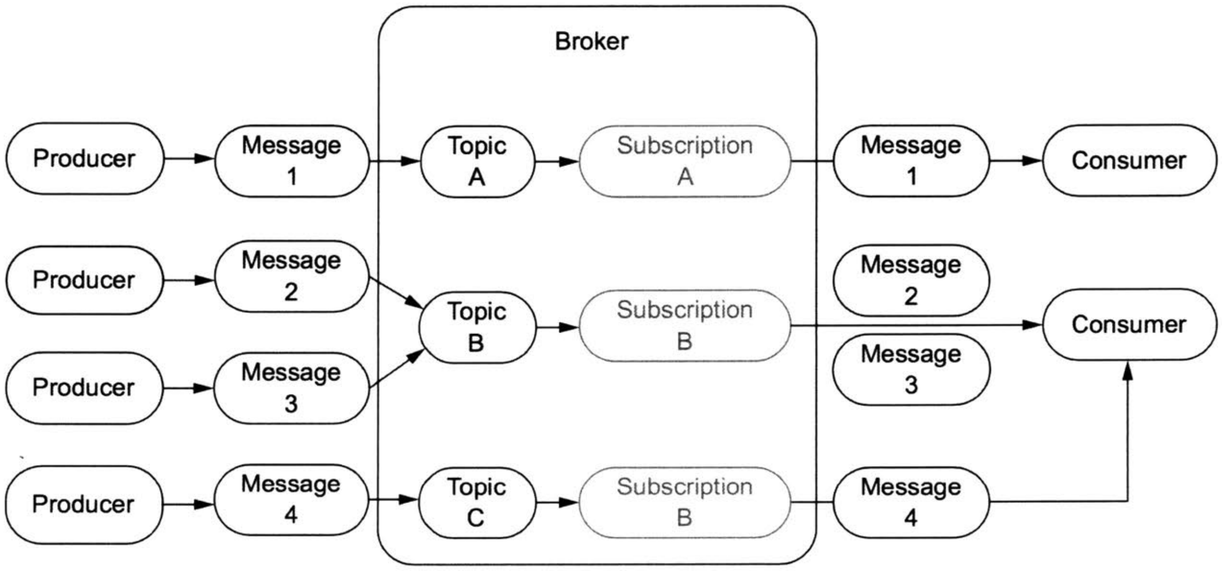


Figure 4-9: Message Broker. Source: (Psaltis, 2017)

4.8.4 Monitoring, Dashboards & Reports

To make sure the platform is operating properly, an organization must have proper tooling in place to continuously monitor all services and components. This includes continuously monitoring the logs generated by the services, the infrastructure's operational parameters.

4.8.5 Security

The platform needs to have security policies that protects the data from being exposed and the servers from being compromised with intrusion or denial of service attacks.

4.9 Conclusion

In this chapter we have presented a reference architecture for an intelligent platform. The architecture combines our survey from the previous chapter in this thesis with our personal experience and research. We presented each component and provided an explanation over the role the components plays in the platform. An organization should be able to take this reference architecture and tailor it to the use cases it is attempting to enable.

It is important to note that even though this reference architecture provides a good overview of the components needed, each organization has to assess if it supports their use cases. This is due to the general nature of reference architectures. Organizations should always assess the requirements they have and treat our architecture as a starting point. They should not assume that it will cover all needed scenarios. Said organizations should also consider the following key decisions when designing their architectures:

- Service level objective of data analysis: Deciding on how fast the data needs to be analyzed after ingestion can have a big impact on the complexity of the overall platform.

- Type of analytical models: The analytical models that the platform needs to support will have a big impact on the complexity of the analysis engines and model management

- Data masking and encryption: The amount of data privacy and encryption needed – which can vary based on the use cases and business domain – can have a big impact on the complexity of the infrastructure needed to ensure the platform is compliant.

- Talent: An organization has to validate that its staff has the correct set of skillsets needed to build such an platform. Specialized skills like data engineering, data science and security architects will be needed to make this platform a success.

In a subsequent chapter, we will develop a cost model that captures the effort needed to develop each of the components detailed in this section.

Chapter 5

Cost Model

5.1 Introduction

In this chapter, we will select a software effort estimation methodology and use it to create a cost model for the reference architecture we previously introduced. Our model will allow an organization to customize it based on the components they decide to implement. It is important to note that we will use the terms effort and cost interchangeably in this section, as effort – presented in units of person-months – can be trivially converted to cost.

5.2 Overview of Effort Estimation Models

There are several published methodologies on software effort estimation. In this section we will provide an overview of the various approaches available, provide a comparison and then select the one we will use to develop the effort model. We found the book "Software Project Effort Estimation" (Trendowicz and Jeffery, 2014) to be an excellent guide in capturing the estimation techniques that exist and their underlying strength and weaknesses. A full taxonomy of software estimation technique can be found in figure 5-1.

5.2.1 Estimation Models

Based on the goals of this thesis, we will be focusing on the following subset of estimation techniques:

- Statistical Regression Analysis: This estimation technique relies on data from previous projects at the organization to develop an estimation model for future projects. Regression coefficients are generated by fitting the regression equation to the historic data. These coefficients are subsequently used to predict the effort of future projects.

- Constructive Cost Model (COCOMO): This estimation technique uses data from software projects across industries to develop a parametric estimation model. The model accepts project size, scaling factors, and effort drivers as inputs from an expert in the organization. It subsequently uses that information to produce an effort estimate.

- Classification & Regression Trees: This estimation technique allows an organization to develop a decision tree based on previously delivered software projects. The nodes in the decision tree constitute decision factors that affect the final effort estimate. The generated tree can then be used for future estimation. Example of a constructed tree can be found in figure 5-2.

- Case-Based Reasoning: This estimation technique relies on analogous historic projects to estimate the effort needed. It assumes that the estimator has a database of historic projects with varying sizes and complexities. When a project needs to be estimated, an analogous is identified from the database

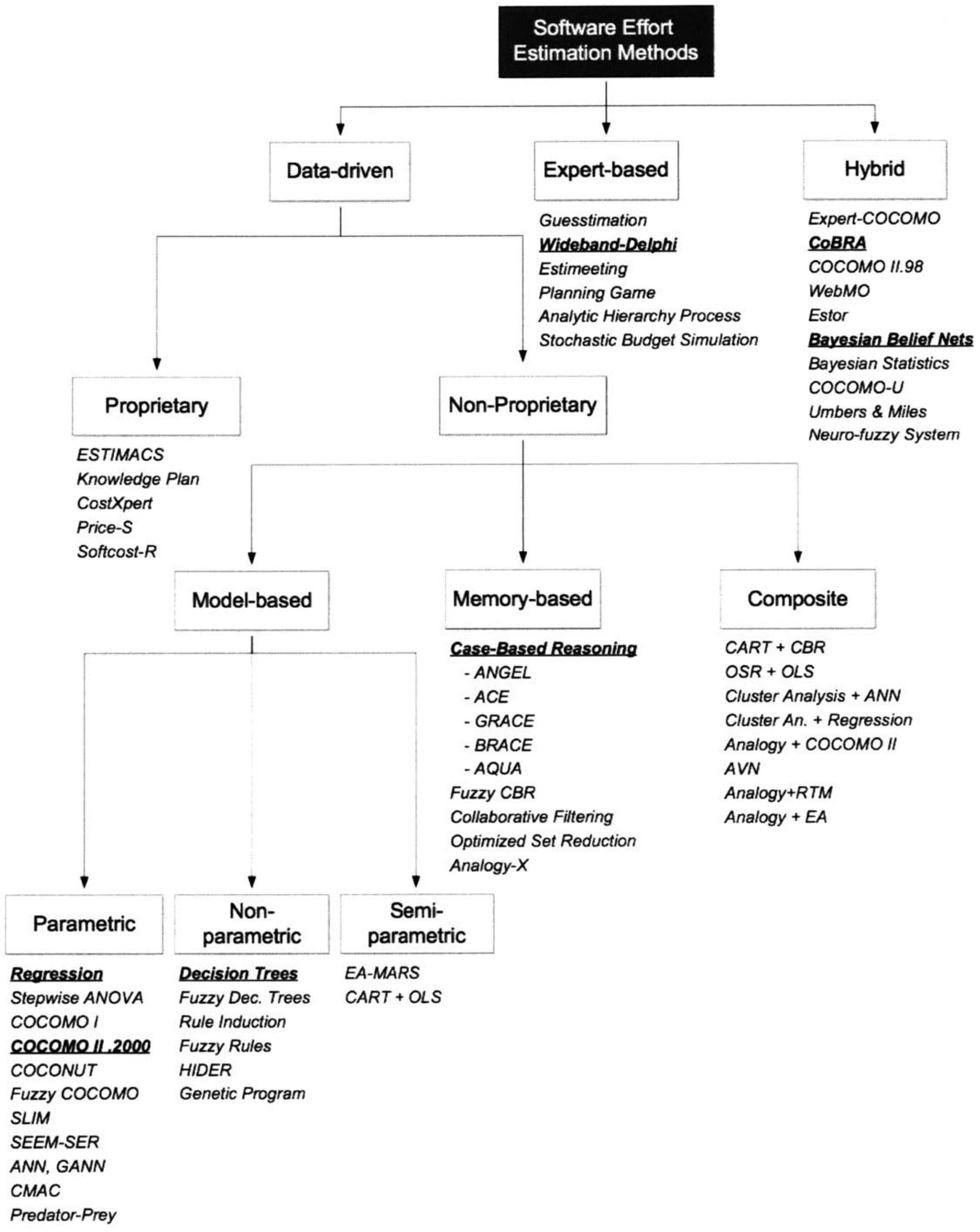


Figure 5-1: Classification of Cost Estimate Techniques. Source: (Trendowicz and Jeffery, 2014)

and the estimates are based on the actual effort that were needed to deliver the historic project.

- Wideband Delphi: This estimation technique relies on several experts providing their best estimate for a task. After they provide their initial estimates, they debate them until they reach consensus.

- Planning Poker: This technique is very similar to the Wideband Delphi technique but is more often used in agile projects. The team debates the complexity of the task at hand during frequent planning sessions throughout the project.

- Bayesian Belief Networks: This technique uses bayesian networks to provide an estimation of the project under consideration. The relationships between the factors affecting the estimation are encoded into a belief network, which is then used to derive the final estimation. An example of a belief network can be seen in figure 5-3.

- CoBRA: This is a hybrid estimation technique that aims at generating a model specific to the organization by combining effort and productivity models with historic project data. An overview of CoBRA can be found in figure 5-4.

5.3 Selecting Estimation Model

The "Software Project Effort Estimation" (Trendowicz and Jeffery, 2014) book details a framework for assessing each estimation techniques to aid in the selection process. We will be following the framework (Figure 5-5) and relying on its 13 decision criteria (table 5.1) to narrow down the estimation methodology to the one that best suits the goals of this thesis. Since we don't yet have access to projects that have followed the reference architecture, we will implement the framework's laid out steps until

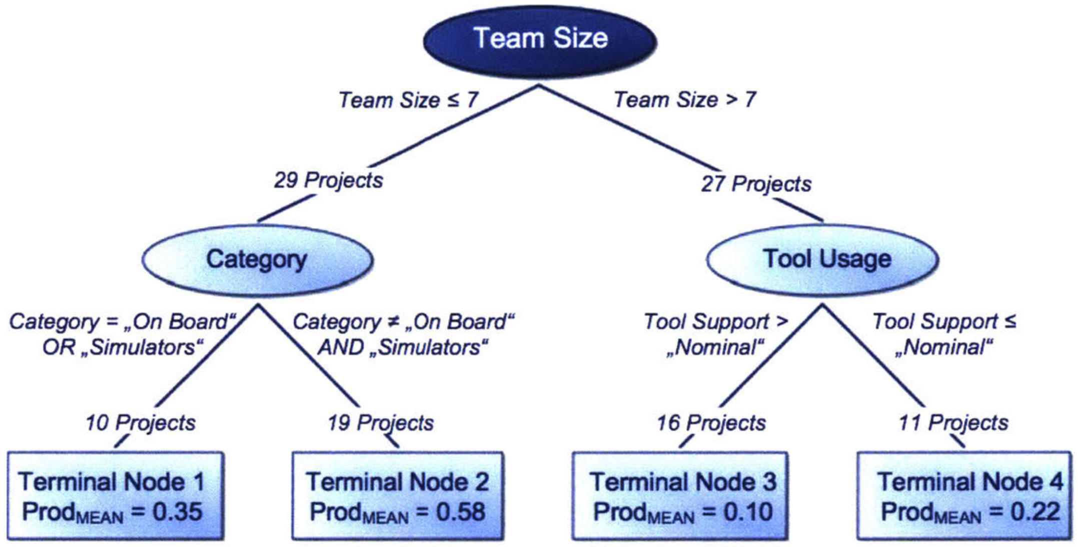

Team Size
Team Size ≤ 7
Team Size > 7
29 Projects
27 Projects
Category
Tool Usage
Category = „On Board" OR „Simulators"
Category ≠ „On Board" AND „Simulators"
Tool Support > „Nominal"
Tool Support ≤ „Nominal"
10 Projects
19 Projects
16 Projects
11 Projects
Terminal Node 1 Prod_MEAN = 0.35
Terminal Node 2 Prod_MEAN = 0.58
Terminal Node 3 Prod_MEAN = 0.10
Terminal Node 4 Prod_MEAN = 0.22

Figure 5-2: Example of a Decision Tree. Source: (Trendowicz and Jeffery, 2014)

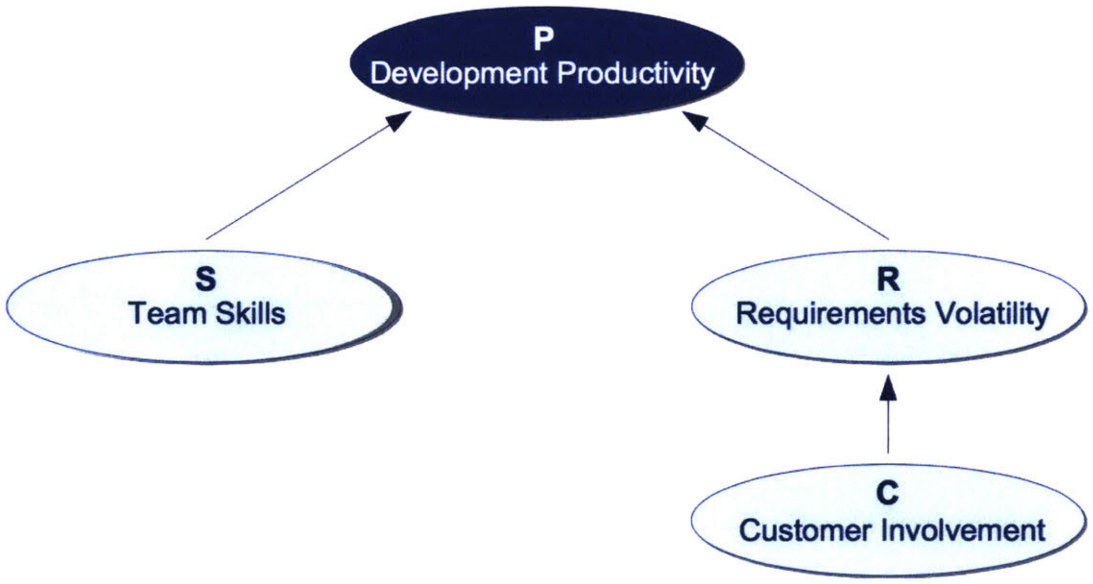


Figure 5-3: Example of a Bayesian Belief Network. Source: (Trendowicz and Jeffery, 2014)

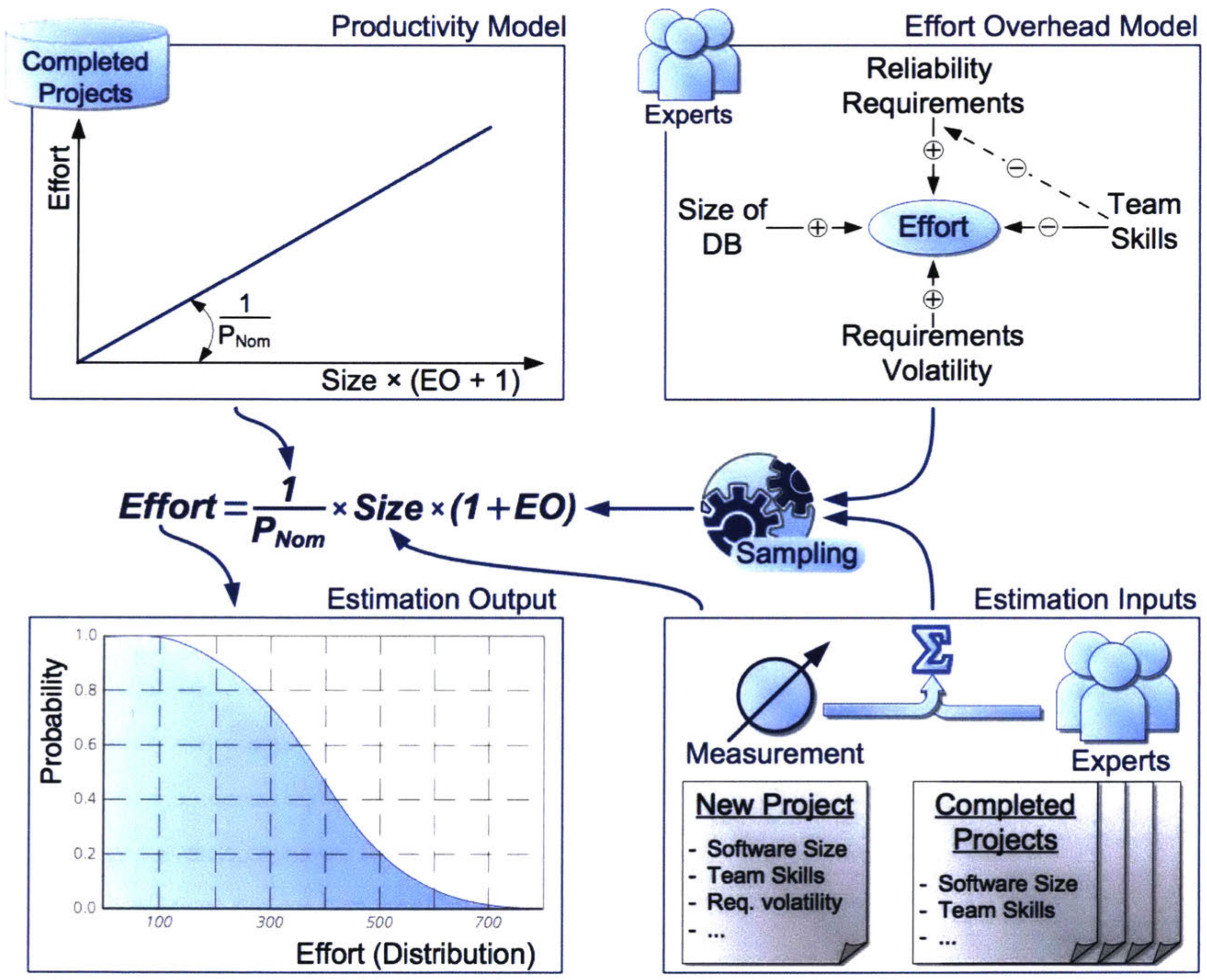


Figure 5-4: Overview of CoBRA. Source: (Trendowicz and Jeffery, 2014)

the "Screen Candidate Effort Estimation Methods" step. We expect future work to extend this thesis to include the subsequent steps in the framework.

5.3.1 Context & Goals

Our intent is to develop an effort model based on the reference architecture we have created. The end-goal is for an organization to estimate the effort needed to deliver such a platform by selecting the desired sub-components and inputting values for the parameters that relate the organization's capabilities. The generated estimate can then be used to determine the amount of work and related cost. This can be important early on in the project launch phase when an initial estimate is needed for stakeholders. Based on this goal we will assign the highest priority to the ability to take input from experts to tweak the model. This will be critical to make sure the estimates are linked to the organization's capabilities. The second priority is the ability to split the estimation over several sub-components and then aggregating them into a single model. This will ensure that customizing a platform by selecting a subset of the sub-components does not break the model.

Since this estimation exercise is done with only the reference architecture, we expect subsequent estimation efforts after the work has been broken down into work streams, and teams have been allocated.

5.3.2 Decision Criteria

To identify the ideal model based on the 13 criteria (Table 5.1), we define our requirements for each criteria in this section and then match those criteria against the strengths and weaknesses of each technique (table 5.2).

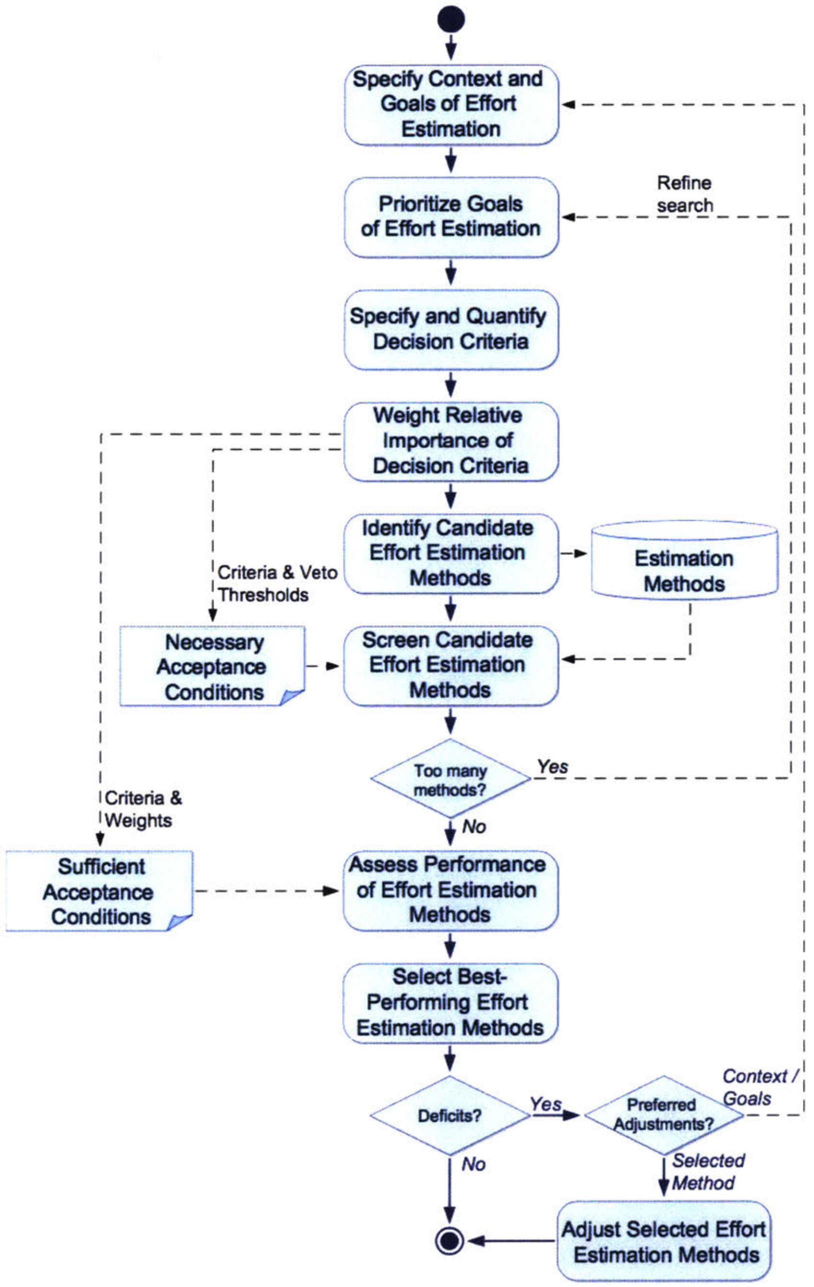

Specify Context and Goals of Effort Estimation
Prioritize Goals of Effort Estimation
Refine search
Specify and Quantify Decision Criteria
Weight Relative Importance of Decision Criteria
Identify Candidate Effort Estimation Methods
Estimation Methods
Criteria & Veto Thresholds
Necessary Acceptance Conditions
Screen Candidate Effort Estimation Methods
Too many methods?
Yes
No
Criteria & Weights
Sufficient Acceptance Conditions
Assess Performance of Effort Estimation Methods
Select Best-Performing Effort Estimation Methods
Deficits?
Yes
No
Preferred Adjustments?
Context / Goals
Selected Method
Adjust Selected Effort Estimation Methods

Figure 5-5: Estimation Technique Selection Process. Source: (Trendowicz and Jeffery, 2014)

Criteria	Description
Expert Involvement	Criterion used to assess the necessary input needed from expert resources to deliver the software estimation
Required Data	Criterion used to assess the dependency on historical data to develop the estimation
Robustness	Criterion for the sensitivity of the estimation to the quality of inputs
Flexibility	Criterion for the flexibility of the estimation technique in relation to the inputs and dependencies
Complexity	Criterion for the complexity of thc underlying theory, algorithms and inputs
Support Level	Criterion for measuring the level of documentation and number of tools available for the estimation technique
Reusability	Criterion for the ease of portability of the developed model for other contexts beyond the initial estimation effort
Predictive Power	Criterion for assessing how close the estimates typically are to the actual project efforts
Informative Power	Criterion for assessing if the estimation method provides additional information beyond the effort estimation
Handling Uncertainty	Criterion for determining the estimation technique's ability to handle uncertainty in inputs and outputs
Comprehensiveness	Criterion refers to the estimation technique's ability to estimate various project activities at varying levels of abstraction
Availability	Criterion to assess the degree the estimation technique can be used for various stages of the software development cycle
Empirical Evidence	Criterion for the amount of evidence available from previous projects

Table 5.1: Software Estimation Model Decision Criteria

Expert Involvement

Since the only information available is the reference architecture, an organization will need to input data into the model to provide information about the organization's ability to deliver such a platform. Therefore, any estimation technique we select needs to accept input from an expert in the organization to perform correct estimation.

Required Data

The estimation technique we choose must not depend only on historic data from an organization. It can rely on industry data, but we will be assuming that an organization either does not have historic data or the data does not apply to this type of project.

Robustness

We expect the model we choose to be somewhat robust to errors in data input. We assume that the value to be supplied to the model might not be of highest quality. We still expect the model to provide adequate estimates. Especially since the estimation effort will be done early on in the project kick off process, where high quality data might not be available.

Flexibility

The model we select should enable users to be flexible with the required parameters. Some of the parameters will not be available, and the user should be able to either fallback to default value, or be able to omit these entirely. We do expect the quality of estimation to drop in these cases, but the model should still provide some useful estimate.

Complexity

Since the estimation process will be done with minimal information, we expect the estimation technique to take simple parameters and not rely on heavy upfront analysis.

Support Level

We expect the estimation technique to have a high level of support. We should be able to understand how it works to calibrate and generate the final model. It should not be a black box and it should lend itself to customization.

Reusability

The final model we will develop will be heavily coupled to the reference architecture we have developed. We do not expect it to be used for any other context. Therefore the underlying estimation model does not have to support reusability.

Predictive Power

It is expected that the accuracy of the model will not be high since the estimation process is being done early on in the project lifecycle. That said, we expect the methodology to generate useful estimates that will be as close to the actual value as needed to make sure the decisions that will be taken are well founded.

Informative Power

The estimation model used should provide a good estimation of effort. Any additional information that the model can offer beyond that will be useful for the project but not a must.

Handling Uncertainty

The methodology we will select has to gracefully handle uncertain inputs. As mentioned in previous criteria, the lack of information early on in the estimation process will create uncertainty around inputs and information. The estimation technique should still deliver useful outputs.

Comprehensiveness

We do not expect the model to be used outside of estimating the software delivery process. Therefore the underlying estimation methodology does not have to support non-technical efforts. Nor should it support varying level of granularity. If a methodology does achieve the above two, that would be considered a plus.

Availability

We expect the model we will develop to be used during the initial stages of the project lifecycle. Once the teams have formed and the work streams have been identified, we expect the project to shift to another estimation technique. That technique will use a finer level of estimation and would rely on more concrete data from the active development efforts.

Empirical Evidence

The underlying estimation methodology we will leverage should be proven to produce good enough estimates at the this early level in the project lifecycle.

5.3.3 Compare Estimation Techniques

To compare the estimation techniques, we will score them based on each decision criteria. Each criteria will be scored from 1 to 5 – the score will represent how closely the estimation method fulfills the requirements for the criteria requirement laid out in previous section, with a score of 5 indicating the requirement is met perfectly.

	Statistical Regression	COCOMO	Classification & Regression Trees	Case Based Reasoning	Wideband Delphi	Planning Poker	Bayesian Belief Network	CoBRA
Expert Involvement	2	4	3	3	1	1	3	3
Required Data	1	5	1	1	5	4	2	1
Robustness	1	3	5	3	1	2	1	1
Flexibility	3	4	4	2	3	4	3	1
Complexity	5	5	2	2	3	4	2	4
Support Level	5	5	5	5	3	4	4	4
Reusability	3	3	3	3	3	2	3	3
Predictive Power	5	4	3	2	3	2	2	4
Informative Power	2	4	3	4	2	3	3	3
Handling Uncertainty	4	3	2	2	4	2	3	4
Comprehensiveness	3	4	3	4	4	3	3	4
Availability	4	4	4	1	4	4	4	3
Empirical Evidence	5	5	5	5	3	1	1	4
Total	**43**	**53**	**43**	**37**	**39**	**36**	**34**	**39**

Table 5.2: Software Estimation Model Decision Criteria

5.3.4 Selected Effort Estimation Method: COCOMO

Based on the analysis shown in table 5.2, Constructive Cost model (COCOMO) emerges as the best underlying estimation method for our model. Specifically, we will be using COCOMO II. COCOMO is a data driven, parametric estimation model that was developed by Barry Boehm in 1981 (Trendowicz and Jeffery, 2014). It was developed using statistical regression on data from multiple organizations. COCOMO II – which was released in 2000 – is a calibrated version of COCOMO. COCOMO accepts two sets of parameters based on the stage of estimation: if the estimation done is post-architecture, 17 effort drivers are used, if pre-design, a reduced set of 7 effort drivers are used instead.

5.4 COCOMO II For Reference Architecture

COCOMO II lends itself excellently to developing our model. Since COCOMO takes the scale factors and the effort drivers as input from the user – and since those factors are organization specific – we simply have to assume that each sub-component will be estimated using the COCOMO model, and the aggregated estimate is the sum of efforts of the sub-components. Since the ideal way to build the sub-components is by breaking them into independent projects that interact via a contract, we will ignore the cost of integrating the components together. Moreover, taking into account that we will estimating during the early stages of project, we will be leveraging the pre-design version of the model:

$$Effort = A \times Size^{E} \times \prod_{i=1}^{7} EM_i \tag{5.1}$$

Where:

$Effort$: Total project effort in person-months

A : Development productivity. Initially A = 2.94

$Size$: Volume of software product measured in lines of code

E : Effect of scale. Detailed in eq. 5.2

EM : Early design effort drivers listed in table 5.3

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j \tag{5.2}$$

Where:

B : Constant initially set to 0.91

SF : Factors that impact the scaling of the project. Detailed in table 5.4

Name	Abbreviation	Description
Personnel capability	PERS	Measurement of the overall capability of the personnel involved in the project
Product reliability and complexity	RCPX	Measurement of the overall complexity of the solution
Platform difficulty	PDIF	Measurement of the complexity of storage and platform volatility
Personnel experience	PREX	Measurement of the overall experience of the personnel and their familiarity with the tools needed to deliver the solution
Facilities	FCIL	Measurement representing the complexity of working across geographically disparate sites
Developed for reusability	RUSE	Measurement representing whether the solution to be developed will be reused in future projects
Required development schedule	SCED	Measurement of the constraints on the expected schedule for delivering the solution

Table 5.3: Early Design Effort Drivers

Name	Abbreviation	Description
Precedentedness	PREC	Measurement representing the organization's familiarity with developing similar projects
Development flexibility	FLEX	Measurement of the flexibility afforded to the development team
Architecture/risk resolution	RESL	Measurement of the risk associated with the solution and any mitigations in place
Team cohesion	TEAM	Measurement of the level of cohesion between the stakeholders, developers and platform end-users
Process maturity	PMAT	Measurement of the process maturity of the organization

Table 5.4: Scaling Factors

Thus, the overall effort model of the reference architecture is :

$$TotalEffort = \sum_{j=1}^{n} SCE_j + IC \tag{5.3}$$

Where:

$TotalEffort$:	The total effort for building the platform in person-months
n :	The total number of components to be built
SCE :	The effort of building the sub components based on eq. 5.1
IC :	The effort of the mandatory infrastructure sub-components (eq. 5.4)

$$IC = S + M + DR + MDR + SEC \tag{5.4}$$

Where:

S :	Estimates for building the servers and networking using eq. 5.1
M :	Estimates for building the message broker using eq. 5.1
DR :	Estimates for building disaster recovery using eq. 5.1
MDR :	Estimates for building the reporting infrastructure using eq. 5.1
SEC :	Estimates for developing the security infrastructure using eq. 5.1

5.5 Estimation Process

The model listed in section 5.4 should provide the information needed for an expert in an organization to estimate the effort – and subsequently the cost – needed to develop the components they select from the reference architecture. The expert will have to execute the following steps:

1. Component Selection: During this phase, the estimators would select the components from the reference architecture that best matches the requirements they have. If any added components is added to the reference architecture, the above model can be easily extended to include them.

2. Size Estimation: For every selected component, the estimators would provide an estimate of the size of the component in kLOC (thousands of line of code). This maps to the "Size" parameter in equation 5.1.

3. Input Effort Driver and Scaling Factor: For each components, the estimators will then enter their subjective value estimate for the scaling factors (Table 5.4) in equation 5.2 and the design effort drivers (Table 5.3) into equation 5.1.

5.5.1 Estimating Effort Drivers and Scaling Factors

The scaling factors and the effort drivers are subjective values that an expert estimator inputs for each component. The design drivers and scaling factors that affect our model are listed in Table 5.3 and Table 5.4 respectively. The estimator will have to enter a rating that represents the significance of the driver or factor under question in relation to the component. The ratings for design effort drivers range from "Extra Low" to "Extra High". Whereas the ratings for the scaling factors range from "Very Low" to "Very High". The numeric values for these ratings can extracted from tables 5.5 and 5.6.

	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
PDIF	-	-	0.87	1.00	1.29	1.81	2.61
PREX	1.59	1.33	1.22	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
RUSE	-	0.81	0.91	1.00	1.11	1.23	-
SCED	1.43	1.14	1.00	1.00	1.00	1.00	-

Table 5.5: Effort Drivers Ratings. Source: (Baik, 2000)

	Very Low	Low	Nominal	High	Very High
PREC	6.20	4.96	3.72	2.48	1.24
FLEX	5.07	4.05	3.04	2.03	1.01
RESL	7.07	5.65	4.24	2.83	1.41
TEAM	5.48	4.38	3.29	2.19	1.10
PMAT	7.80	6.24	4.68	3.12	1.56

Table 5.6: Scaling Factor Ratings. Source: (Baik, 2000)

5.6 Conclusion

To select an estimation model to base our cost model on, we identified 13 criteria that the estimation technique should satisfy. Based on those criteria, COCOMO II emerged as the ideal estimation model. We then developed the cost model by applying the COCOMO II over the components in the reference architecture. We identified the scaling factors and design effort drivers that will impact the estimation process, and listed the values needed during estimation. With the information listed here, an estimator can develop an initial cost estimate for the architecture they wish to develop.

Chapter 6

Conclusions

In this thesis we have developed a reference architecture that can be used to build intelligent platforms. These platforms can enable use cases that leverage Machine Learning and Artificial Intelligence. We have also selected the most adequate software estimation methodology for performing a cost estimate for the platform early in the project kick-off process. This work should enable an organization looking into building an intelligent platform to understand the components that it needs to build and their respective cost estimates. However, we believe more work can be done in this area :

- Work is needed to validate the cost model. This can be achieved by tracking a build of an intelligent platform from early estimation to delivery, followed by validating the estimates against the actual values. This has the potential to expand the model to include any estimation factors that were missed.

- We have leveraged the COCOMO II model to build our cost model. COCOMO was built before projects relied heavily on Machine Learning and AI models for execution. Building these models is a complex undertaking that is cost and time intensive. Training this models is often more of an art than a science, as the data scientists experiment with tweaking their models for weeks at a time before attaining acceptable accuracies. Further work needs to be done to understand if COCOMO estimates empirically hold for projects that depend heavily on this type of model development.

- The reference architecture does not detail the latency of data processing needed. Depending on use cases, some platforms have strict real-time data processing requirements that mandate a certain architectural decisions to achieve the time constraints. Further work can expand the architecture to measure the latency of data processing based on the sub-components configuration, and detail optimization techniques.

Bibliography

Alpert, J. and Hajaj, N. (2008). We knew the web was big...

Anonymous (1865). The story of life insurance: Chapter 1. john graunt, citizen and haberdasher. *The Railway News and Joint Stock Journal*, pages 439–441.

Anonymous (2012). Altior's altrastar - hadoop storage accelerator and optimizer now certified on cdh4 (cloudera's distribution including apache hadoop version 4).

Anonymous (2017a). Ec2 instances.

Anonymous (2017b). Making sense of the census: Hollerith's punched card solution.

Baik, J. (2000). *COCOMO II Model Definition Manual.* Center for Software Engineering.

Bazar, C. and Iosif, C. S. (2014). The transition from rdbms to nosql. a comparative analysis of three popular non-relational solutions: Cassandra, mongodb and couchbase. *Database Systems Journal*, 5(2):49 – 59.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* Information science and statistics. New York : Springer, c2006.

Black, B. (2009). Ec2 origins.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422 – 426.

Bryson, S., Kenwright, D., Cox, M., Ellsworth, D., and Haimes, R. (1999). Visually exploring gigabyte data sets in real time. *Commun. ACM*, 42(8):82–90.

Bughin, J. (2016). Big data: Getting a better read on performance.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., and Bone, M. (2010). The concept of reference architectures. *Systems Engineering*, 13(1):14 – 27.

Codd, E. (1983). A relational model of data for large shared data banks. *Communications of the ACM*, 26(1):64 – 69.

Dean, J. and Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107 – 113.

Dennis J, M. (2016). Analyzing and interpreting 'imperfect' big data in the 1600s. *Big Data & Society, Vol 3, Iss 1 (2016)*, (1).

Flajolet, P., Fusy, Ã., Gandouet, O., and Meunier, F. (2007). Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *2007 Conference on Analysis of Algorithms, AofA 07*, Discrete Math. Theor. Comput. Sci. Proc., AH.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM.

Gillis, T. (2015). Cost wars: Data center vs. public cloud.

Goda, K. and Kitsuregawa, M. (2012). The history of storage systems. *Proceedings of the IEEE*, 100(Special Cen):1433 – 1440.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning.* Adaptive computation and machine learning series. Cambridge, MA : MIT Press, [2016].

Gramatica, R. and Pickering, R. (2017). Yewno: an ai-driven path to a knowledge-based future. *Insights: the UKSG journal*, 30(2):107 – 111.

Graunt, J. (1676). *Natural and political observations mentioned in a following index, and made upon the bills of mortality.*

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning.

Isaacson, W. (2014). *The innovators.* New York, NY : Simon & Schuster, 2014.

Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Prentice Hall series in artificial intelligence. Upper Saddle River, N.J. : Pearson Prentice Hall, c2009.

Kleppmann, M. (2017). *Designing data-intensive applications : the big ideas behind reliable, scalable, and maintainable systems.* Sebastopol, CA : O'Reilly Media, 2017.

Leavitt, N. (2009). Complex-event processing poised for growth. *Computer*, 42(4):17 – 20.

Lee, G., of Technology. Department of Electrical Engineering, M. I., and Computer, S. (2015). *Hierarchical planning for multi-contact non-prehensile manipulation.*

Levy, S. (2011). *In the plex : how Google thinks, works, and shapes our lives.* New York : Simon & Schuster, c2011.

Lewin, C. G. (2004). 'graunt, john (1620-1674)', oxford dictionary of national biography, oxford university press.

Luhn, H. P. (1958). A business intelligence system. *International Business Machines Corporation. Journal of Research and Development*, 2:314.

McCarthy, J., Minsky, M. L., and Rochester, N. (2006). A proposal for the dartmouth summer research project on artificial intelligence. *AI Magazine*, 27(4):12 – 14.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115.

Meehan, J., Tatbul, N., and Du, J. (2017). Data ingestion for the connected world.

Mellor, C. (2010). Google's storage strategy.

Microsoft (n.d). Understanding microsoft big data solutions.

Morabia, A. (n.d.). Epidemiology's 350th anniversary: 1662-2012. *EPIDEMIOLOGY*, 24(2):179 – 183.

Muller, B. and Reinhardt, J. (1990). *Neural networks : an introduction.* Physics of Neural Networks. Berlin : Springer, 1990.

Munson, J. H. (1968). Experiments in the recognition of hand-printed text, part i. *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part II*, page 1125.

Mysore, D., Khupat, S., and Jain, S. (2013). Big data architecture and patterns: Understanding the architectural layers of a big data solution.

Nilsson, N. J. (2010). *The quest for artificial intelligence : a history of ideas and achievements.* Cambridge ; New York : Cambridge University Press, 2010.

NIST Big Data Public Working Group (2015a). Nist big data interoperability framework: Volume 3, use cases and general requirements. Technical report, NIST.

NIST Big Data Public Working Group (2015b). Nist big data interoperability framework: Volume 5, architectures white paper survey. Technical report, NIST.

NIST Big Data Public Working Group (2015c). Nist big data interoperability framework: Volume 6, reference architecture. Technical report, NIST.

NTT Data Corporation (2015). Big data reference architecture.

Psaltis, A. G. (2017). *Streaming Data.* Manning Publication.

Randell, B. (1982). *The Origins of digital computers : selected papers.* Texts and monographs in computer science. Berlin ; New York : Springer-Verlag, 1982.

Reinsel, D., Gantz, J., and Rydning, J. (2017). White paper: Data age 2025 : The evolution of data to life-critical. Technical report, International Data Corporation, Framingham, Massachusetts.

Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton.

Rubinfeld, D. L. and Gal, M. S. (2017). Access barriers to big data. *Arizona Law Review*, 59(2):339 – 381.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1):206 – 226.

Shankland, S. (2009). Google uncloaks once-secret server.

Szeliski, R. (2011). *Computer vision. [electronic resource] : algorithms and applications.* Texts in computer science. New York ; London : Springer, 2011.

Trendowicz, A. and Jeffery, R. (2014). *Software project effort estimation : foundations and best practice guidelines for success.* Cham : Springer, [2014].