Big Data Techniques for Predictive Business Intelligence

Keith Gutfreund
Elsevier Labs, Cambridge, MA
Email: k.gutfreund@elsevier.com

***Abstract*—Using Apache Spark™, Natural Language Processing (NLP), and text mining, the author's team analyzed enterprise Customer Relationship Management (CRM) data to predict and combat premature sales attrition. NLP techniques were employed to identify critical patterns in unstructured CRM data fields. The Apache Spark big data engine and the Databricks™ big data software as a service (SaaS) platform were then used to efficiently locate evidence of these patterns within large amounts of unstructured CRM data. New business reports are being generated from these patterns so that in-house support staff may proactively engage with customers. Future work will focus on automating these techniques to identify new problems and to quickly spot emerging trends.**

***Index Terms*—big data, apache spark, Natural Language Processing (NLP), Customer Relationship Management (CRM) software**

I. Introduction

Like most large enterprises, Elsevier uses Customer Relationship Management (CRM) software to manage multi-channel communications between its customers and its in-house support teams.

The customer data are collected from electronic forms (email, web) as well as from transcribed voice conversations. The data are then manually annotated and classified/categorized into a small number of fixed-value fields by trained customer service representatives. Both the classified/categorized data fields, as well as the large amount of free, unstructured text, are incorporated into an Oracle relational database.

Elsevier has thousands of customers and voluminous CRM data, with new customer data arriving continuously. The Oracle database provides structured query search capabilities, although due to the large amount of data, this is limited to off-line batch mode processing of the structured data fields indexed by the database software.

The author's team is using big data engines, natural language processing (NLP), and text mining to improve the system in two important ways:

1. Big Data techniques provide interactive access to the CRM data for search, experimentation, and analytics.
2. The large amount of unstructured data is mined to yield new analytics and predictive business intelligence.

Big Data computation engines, NLP techniques, and text-mining algorithms allow us to discover new customer problems, identify trends, and proactively combat sales attrition. For example, we can discover problems hidden in transcribed free text conversations with phrases like: "I'm having trouble accessing resource X" or "My login isn't working today". Further, we can preemptively combat sales attrition by scanning conversations for patterns like, "I'm very unhappy with product (or feature or service) Y", and then proactively engaging with the unhappy customers.

Our goal was to examine raw customer interactions using tools and techniques that are not part of contemporary CRM packages, namely Big Data computation engines and natural language processing algorithms. We hoped to accomplish two objectives:

1. Increase the amount of structured information that could be used for reporting, and
2. Develop a trend spotting method to identify new problems early.

A. The Enterprise CRM Dataset

Elsevier customer data is stored in the Oracle RightNow CRM software product [1]. RightNow provides a repository for storing email, web and telephone customer interactions, and the repository may be queried interactively, using Oracle RightNow tools and interfaces, as well as programmatically, using custom database access software. The data is categorized upon entry and data extracts from the repository are provided to in-house analysts. Additionally, a variety of reports are generated from this data. The repository holds millions of customer interaction records.

We experimented with a RightNow data sample containing approximately 231k customer interaction records. Each record in the dataset contained the fields shown in Table I. Three fields are of particular interest to this paper. The Disposition Level 1 and Disposition Level 2 fields provide a two-level categorization of the customer incident, and a large, unstructured text field contains the historical communications between the customer and the support staff, as well as between individual support staff members during an incident. At

Manuscript received October 1, 2015; revised January 21, 2016.

doi: 10.18178/joams.5.2.158-163

the beginning of this project, the large text field was not used for any analytical purpose.

TABLE I. RIGHTNOW RAW DATA FIELDS

Field	Example
ID	141031-006570
Date (original)	31/10/2014 10.48 PM
Date (normalized)	2014-10-31T00:00:00.000
Date (closed)	2014-10-31T00:00:00.000
Channel ID	Web
Product	Scopus
Subject	The system doesn't seem to be working
Contact type	Employee of academic institute
Flag	0
Queue	Europe
Mailbox	NL Info
Status	Solved – No Response
Text	The system doesn't seem to be working when I try to log on and I have a deadline soon and I need to reference my sources…
Disposition Level1	Authentication
Disposition Level2	Cannot login

B. Big Data Tools

A traditional non-Big Data approach to analyzing large datasets with millions of data records and unstructured text fields typically requires machines with powerful processors, large disks, and vast amounts of random access memory. This approach is costly in terms of hard ware and it does not scale beyond the largest machines that can be acquired for the task. Alternatively, contemporary Big Data techniques allow datasets to be distributed over many modestly sized machines and also allows for parallel processing on the distributed data. Apache™ Hadoop®(http://hadoop.apache.org) is an open-source Big Data software framework for distributed storage and processing of large data collections - it is the most well-known Big Data tool. Hadoop provides for a distributed disk-based file system and it supports the well-known MapReduce programming interface on this data.

Jeffery Dean and Sanjay Ghemawat created MapReduce at Google in the early 2000's and released it in 2004 [2], [3]. MapReduce is a combination of a software programming model and an implementation for processing and generating large data sets. The MapReduce programming library hid the "… messy details of parallelization, fault-tolerance, data distribution and load balancing." [2], [3]. Simple MapReduce constructs yielded solutions that were both efficient and scalable and permitted automatic parallelization and distributed data processing.

In 2004 Dean and Ghemawat cited examples of interesting programs easily expressed in MapReduce; we find that those are still relevant today. Two of these problems are:

- Web document (word, frequency) pair computations to determine the most frequent and/or important words appearing in a document
- Distributed sorting of key, record pairs

We make use of both of these programming examples in this paper.

MapReduce itself does not define a new programming model; rather, it is based on functional programming languages like Common Lisp [4] along with methods and message-passing systems that were first developed in the early 1990's. At the First CRPC Workshop on Standards for Message Passing in a Distributed Memory Environment in 1992 [5], participants discussed the need of a system to address the most difficult aspects of distributed memory applications, viz.:

1. Devising a correct parallel program
2. Optimizing the program code for efficient and scalable performance.

These same functional programming ideas and message passing techniques are intrinsic to MapReduce.

Following upon Dean and Ghemawat's MapReduce work, a first non-Google implementation of MapReduce, called Hadoop, was created in 2005 by Doug Cutting and Mike Cafarella. Hadoop was originally part of the Nutch search engine project [6].The Hadoop package includes both a distributed file system called the Hadoop Distributed File System (HDFS) and the MapReduce software library. Since 2006, Hadoop has been widely used for a variety of distributed computing problems. See [7], [8] for examples.

While Hadoop works well for some problems, it suffers from some significant deficiencies.

- The MapReduce programming paradigm is not a good fit for many programming abstractions
- Hadoop MapReduce is made to run on distributed data stored on disk, the Hadoop Distributed File System (HDFS), and not in RAM

These deficiencies eventually led to a new solution for Big Data processing, namely Apache Spark™.

Apache Spark is the newestopen-source software framework for big data [9]. Spark differs from Hadoop in two important ways:

1. Spark's distributed data collection is stored in RAM, providing for a much faster processing engine.
2. Spark has built-in programming abstractions and libraries that make it a natural fit for applications using Structured Query Language (SQL), Extract, Transform Load (ETL) algorithms, and machine learning [10].

Apache Spark has several features and programming abstractions that were useful for this paper:

In-memory RDDs: Spark distributes data across worker machines using fault-tolerant resilient distributed datasets (RDDs). Unlike Hadoop's HDFS, Spark's RDDs will use RAM but will spill data to disk as necessary. We were able to store much of our data in memory, which dramatically improved processing time. The following

two lines of Python code show loading a file into an RDD distributed across multiple worker machines and then printing the first 3 lines:

```
>>>inputRDD = sc.textFile("/mnt/data.txt.gz")
>>>inputRDD.take(3)

Results:
- [u"141031-006570\t31/10/2014 10.48 PM\t2014-
10…
- 014 04.52 PM\tWeb\tScopus\tThe system doesn't
seem …
- 4.52 PM\tWeb\tScopus\tThe system doesn't seem
to be working…
```

The "sc" above is the Spark context and represents the main program entry point for Spark functionality. The "/mnt/…" path is the data path to the file to be loaded.

Lazy evaluation: Both Hadoop and Spark solve complex tasks by cascading multiple MapReduce operations. Spark's lazy evaluation postpones actual code execution, thus permitting Spark to optimize the order of evaluation. Below is a simple example demonstrating Spark's lazy evaluation:

```
>>> inputRDD = sc.textFile("/mnt/
ScopusIncidentData.txt.gz")
Command took 0.08s

>>> inputRDD.count()
Out[1]: 231671
Command took 2.82s
```

Above, Spark reports that the loading of a text file containing 231,671 lines into RAM from an external file system took only 0.08 seconds, whereas counting the number of lines in the file took 2.82 seconds. Spark delayed reading the text file until that data was actually required by the count () operation.

Direct access to distributed data: Spark permits operations directly on RDDs, and provides abstractions to view RDDs as relational databases. Spark provides SQL operations (like SELECT * from table) as shown below:

```
>>>rows = sqlContext.sql('select * from
trouble_table').take(2)
>>> for x in rows:
  print x
Row(Message=u'onsider the ethical plagiarism
[Reference: 141031-003226] We are escalating …
Row(Message=u'onsider the ethical plagiarism
[Reference: 141031-003226] Incident No: 141031-
003226 Mail forwarded from …
Command took 5.03s
```

Above, we used a Spark SQL Select * command to read and display two rows from an RDD.

In this work we used Spark to deploy our data across multiple machines using Spark's resilient distributed datasets (RDDs), we employed various MapReduce operations, and we used SQL abstractions and other operations directly on distributed data.

Lastly, one significant challenge for both Hadoop and Spark is the lack of an out-of-the-box interactive, cloud-based, big data framework. Lacking such a framework solution providers are required to repeatedly size and design the hardware infrastructure needed to manage (start, stop, monitor) the multiple hardware components within a MapReduce implementation. This is a lot of work. Fortunately, Databricks (http://databricks.com), a company formed by the Apache Spark inventors at the University of California, Berkeley, provides an interactive shell over a cloud-based Spark implementation that handles much of this work. Databricks manages the various hardware components of a MapReduce implementation, it provides an interactive shell for experimentation and development, and it provides tools to monitor and debug performance. Our Databricks implementation reported in this paper includesboth Scala and Python programs running Apache Spark on Amazon cloud instances.

The Databricks shell is a notebook interface for interacting with Spark. The Databricks notebook allows the user to intermix basic markup display with Python, Scala, R, and SQL code. In Fig. 1 below, the Databricks notebook starts with a title "Spark Scopus Incident Program," written in markup. Next, Python Spark code loads a CRM text file into an RDDand then uses a filter transformation to create a new RDD without the first header line in the data file.

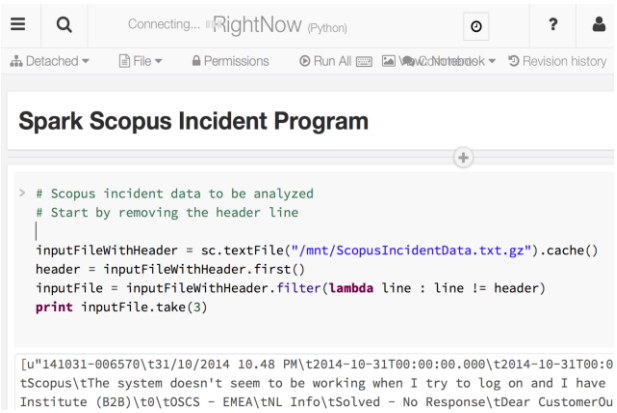

Figure 1. Databricks Notebook screen shot with mark up and code.

C. *Natural Language Processing*

Natural Language Processing (NLP) is a blend of computer science, artificial intelligence, and computational linguistics [11]. NLP dates back to the 1950s and 1960s with early experiments in language understanding [12] and translation programs [13].

We employed a couple different NLP techniques in this project. First, we implemented a concordancer [14] to extract sentences containing key phrases and to examine words preceding and following those key phrases. As an example, some selected output from our concordancer for the key phrase "trouble with" is shown in Table II.

TABLE II. CONCORDANCER OUTPUT

ID	Output
141002-001480	that you are having trouble with your purchase. In order to
140929-007053	scussed I am having trouble with advanced author searches
140929-003433	hear you are having trouble with access to Science Direct.

Next, we looked at single-word (unigram), two-word (bigram) and three-word (trigram) phrases preceding and following the key phrases supplied to the concordancer. In text analysis applications, n-grams language models (unigram, bigram, trigram …) along with Markov models are used to probabilistically predict words in a sequence [15]. In our application we are examining the different n-grams for their significance in identifying important trends.

The following Python code fragment is used to return a list of n-grams found within a body of text. Given a concordancer retrieval, we would invoke the Python makeNGrams () function twice, once for the words preceding the key phrase and once for the words following the key phrase.

```
# Make n-grams
#   n - size of n-gram
#   text - string to break into n-grams
#
# Return list of n-gram strings
import re
def makeNGrams(n,text):
  # Split the words, discard non-letters
  words = text.split(" ")
  filtered_words=[]
  for w in words:
    if re.match("[a-zA-Z]+", w):
      filtered_words.append(w)
  words = filtered_words

  # Convert n-gram to space separated words
  ngrams = []
  for i in xrange(len(words) -(n-1)):
    gram = ''
    for j in xrange(n):
      gram += words[i+j] + ' '

    # Returning list of ngram strings
    ngrams.append(gram.strip())

  return ngrams
```

Frequently occurring n-grams were then further investigated for trend analysis.

II. Approach

Our team analyzed the unstructured text field in each data record and with the goal of identifying patterns and the early discovery of new customer problems. We typically started with an intuition like: "I'm having a problem with feature X" and then enhanced it to: "I'm having trouble with feature X" and "I'm having difficulty with feature X". We employed software regular expressions to assist with the pattern matching, so that the above pattern would be similar to: "I'm having (problem*|trouble|difficult*) (with|doing) X". The vertical bar '|' indicates the logical disjunction ("OR") of multiple choices, e.g. the regular expression "(with|doing)" matches sentences containing the word "with" or the word "doing". The asterisk '*' matches zero or more letters, e.g. "difficult*" matches the words "difficulty "and "difficulties". The "\d" matches a digit, e.g. "IE\d" would match IE7, IE8, or IE9. After applying these regular expressions against the text field in all records, we then count the occurrences of the different values of X and display the most frequently occurring values. These matches would be what our users were most often reporting as problems, as troublesome, or as difficult.

TABLE III. Sample Patterns to Identify New Problems

Browser Issues
browser | firefox | safari | chrome | internet explorer | IE\d
Legal, Ethical Issues
plagiarize | plagiarism
cheat | cheating
legal (action | proceeding)
Products
Reaxys
Scopus
ScienceDirect
SciVal
SciVerse
Cut off Access
authentication
restore
remove
suspen(d | sion)
delete(d)
deactivate
disable
excessive usage

TABLE IV. Sample Patterns to Identify Trends

Trend: Broken or Not Working
(not working now) | (no longer working) | (was working)
((does not)|doesn't) work
(is | are | now) broken
(can't | can not | can no longer)
having (trouble | difficulty | problem) with
(is now) | (has been) | (is still) unavailable
(serious | substantial | significant | real) (trouble | difficulty | problem)
Trend: Customer Satisfaction
not happy
(very | extremely | serious) (unhappy | upset | dissatisfied)
Trend: Customer Asking for Help for Urgent Matters
in proximity of "urgent" or "serious": (please help) | (please fix) | (need help)

Table III shows some sample patterns we employed to identify new problems in the data. Here, one interesting pattern we discovered arose from customers reporting legal and ethical issues, including plagiarism and cheating. A second pattern attempted to identify additional products that were involved in a customer complaint. A third pattern attempted to classify the reasons why a customer's account was deactivated. For example, excessive usage might indicate users crawling the product for content. These problems had not previously been tracked.

Table IV shows some sample patterns we employed to identify new trends in the data. Our hope is that these patterns will help us to quickly identify and address emerging issues leading to customer dissatisfaction. For example, features that suddenly stopped working could

be identified in text matching the expression: *(X not working now)* | *(X no longer working)* | *(X was working)*. Customer dissatisfaction could be detected in text matching the expression: *(very | extremely | serious)* followed by *(unhappy | upset | dissatisfied)*

Once we have selected a promising pattern, we then use the concordance and our n-gram software to identify key values for a trend or issue. The Databricks Spark platform provides an interactive platform that facilitates easy experimentation, structured query language (SQL) lookup operations, and some basic visualizations.

Fig. 2 shows a Databricks visualization of the distribution of Channel values in reported incidents over different time periods.

Fig. 3 shows a Databricks visualization of the final disposition of customer problems reported over time.

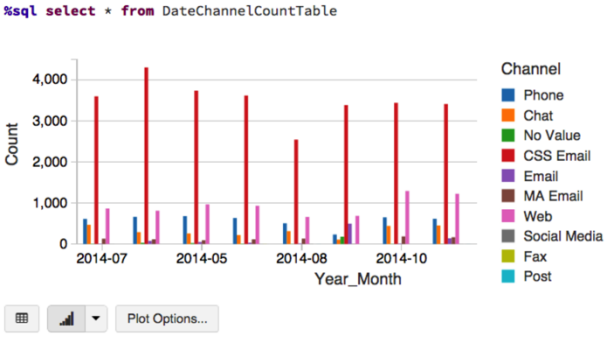

Figure 2. Databricks visualization of reporting channel over time.

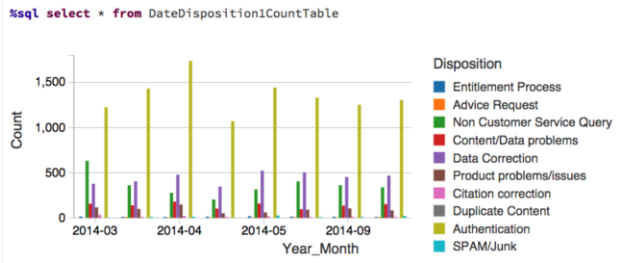

Figure 3. Databricks visualization of problem disposition over time.

III. Next Steps

Now that we have new sets of patterns to identify problems and trends, we are in the process of migrating these to the RightNow CRM reporting system as standing queries and prepared statements. These will then be included in our regular analysis.

One powerful next step is to automate both the ingestion and the analysis of CRM data. As new customer reports are transcribed and as customer emails arrive, new data records are automatically added to the existing CRM system.

Nevertheless, appending new data records to a Spark dataset requires a slightly different approach. Spark's Resilient Distributed Datasets (RDDs) are immutable–once created, an RDD is not modifiable. This prevents us from appending new data to an existing RDD. Nevertheless, if we treat the newly arriving data as a data stream; Spark provides a programming engine and abstraction that transforms a data stream into a sequence of RDDs, as show in Fig. 4. Spark calls this abstraction a discretized stream (DStream).

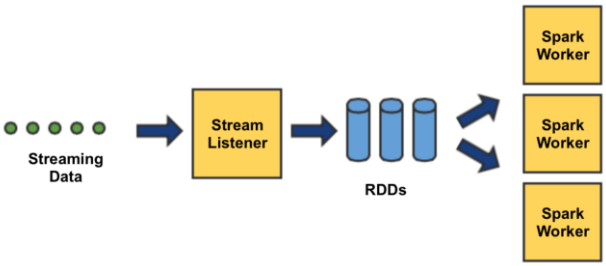

Figure 4. Spark Discretized Stream (DStream) as batch RDDs.

The sequence of RDDs that make up a DStream may be processed individually as shown in Fig. 4 or a DStream may be combined with an RDD. The Spark Python demonstration code below shows a map-reduce word frequency computation being carried out on each arriving batch of CRM data arriving at a TCP socket.

```
from pyspark.streaming import StreamingContext

# Read and process CRM data stream every 60sec
stream_sc = StreamingContext(sc, 60)
records= stream_sc.socketTextStream(host,port)

# Separate tab-delimited record into fields
fieldsRDD = records.map(lambda x: x.split('\t'))

# Operate on the text field of CRM data
textRDD = fieldsRDD.map(lambda x: x[10])

# RDD of the words in the text field
wordsRDD=textRDD.flatMap(lambda x: x.split(' '))

# Frequency of words appearing in this batch RDD
pairs = wordsRDD.map(lambda x: (x,1))
word_counts = pairs.reduceByKey(lambda x,y: x+y)

# Print some word-count pairs in the batch
word_counts.pprint()
```

IV. Conclusion

Customer Relationship Management (CRM) data may provide a trove of valuable information. Two key obstacles to tapping this information are 1) that the data are frequently unstructured, and 2) that there is so much data to analyze. This paper demonstrated that Natural Language Processing (NLP) techniques could be used in Big Data platforms like Spark to mine the unstructured textual data and discover interesting and useful patterns for business intelligence. Much work remains to be done, particularly to automate the data collection and analysis.

Acknowledgment

The author thanks Matt Cumberlidge and Paul Whitehouse from Elsevier's eBusiness and Information & Intelligence departments, and Ron Daniel and Darin McBeath from Elsevier Labs for their assistance on this project. The author thanks Myrna Gutfreund, Olivia Gutfreund, Maxine & Sander Gutfreund, Les Gutfreund and Lynn Malkin for their patience, love and support.

REFERENCES

[1] Oracle Corp. (2012). RightNow Web Experience Data Sheet. [Online]. Available: http://www.oracle.com/us/products/applications/rightnow/overview/index.html

[2] J. Dean and S. Ghemawat, "MapReduce: A simplified data processing on large clusters," in *Proc. the 6th Symposium on Operating Systems Design & Implementation*, San Francisco, CA, Dec. 6, 2004.

[3] J. Dean and S. Ghemawat, "MapReduce: A simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[4] American National Standards Institute, *Common Lisp*, ANSI X3.226:1994, ANSI INCITS 226-1994.

[5] D. Walker, "Standards for message passing in a distributed memory environment," *Technical Report TM-12147*, Oak Ridge National Laboratory, Oak Ridge, TN, Aug. 1992.

[6] T. White, *Hadoop: The Definitive Guide*, 3rd ed. Sebastopol, CA: O'Reilly, 2012, pp. 11-12.

[7] M. Bhandarkar, "MapReduce programming with Apache Hadoop," in *Proc. 2010 IEEE International Symposium on Parallel & Distributed Processing*, Atlanta, GA, Apr. 2010.

[8] M. Grover, T. Malaska, J. Seidman, and G. Shapira, *Hadoop Application Architectures*,1st ed. Sebastopol, CA: O'Reilly, 2015.

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. HotCloud 2010, 2nd USENIX Workshop on Hot Topics in Cloud Computing*, Boston, MA, 2010.

[10] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark, Lightning-Fast Big Data Analysis*, 1st ed. Sebastopol, CA: O'Reilly, 2015.

[11] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, 2nd ed. Cambridge, MA: MIT Press, 2000.

[12] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433-460, Oct. 1950.

[13] (Jan. 8, 1954). IBM Press Release. 701 Translator. [Online]. Available: http://www-03.ibm.com/ibm/history/exhibits/701/701_translator.html

[14] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, 2nd ed. Cambridge, MA: MIT Press, 2000, ch. 1.4.5, pp. 31-34.

[15] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2008.

Keith Gutfreund is Disruptive Technologies Director at Elsevier Labs, Elsevier B.V. Mr. Gutfreund received his Bachelor of Science in Electrical and Computer Engineering from the University of Michigan, Ann Arbor, MI USA. At Elsevier Labs, he is involved in big data, natural language processing and search engine design and application projects. His earlier academia positions include Lecturer at The National Engineering School of Tunis, Tunisia, at Framingham State University, Framingham, MA USA, and at Lawrence Technological University, Southfield, MI USA. Mr. Gutfreund's team entry at the Fifth International Workshop on Human-Computer Interaction and Information Retrieval (HCIR) at Google Headquarters, Mountain View, CA USA, earned second place in the 2011 HCIR search engine challenge. In 2008, Mr. Gutfreundco-developed a 3D brain simulation program that was featured on the United States National Public Radio show "All Things Considered." Mr. Gutfreund published "Internet Content Filtering Protocol," an Internet Engineering Task Force (IETF) Draft Standard as well as papers on both hardware and software engineering. In 2001, Mr. Gutfreund's team was granted a patent for work at Alta Vista.