Building Software That Captures Better Data

Daniel Casale

dcasale@gmail.com

Dan Elbaum

dan@danelbaum.com

Abstract

The data that we collect from end-users can be just as valuable as the functionality that we deliver to end-users. But do you think through data requirements as carefully as you think through functional requirements? Given the pressure to deliver quality functionality, it's easy to forget that data quality is another important dimension of software quality, and that data requirements are a distinct class of requirements worthy of analysis in their own right. With the costs of data collection falling and the power of data analysis tools climbing, getting data requirements right is growing in importance.

So, how do we get data requirements right? How do we identify what data we need to acquire now so that we have the information we need to make good decisions later? How should we store that data in order to ensure that it supports analytics? This paper will describe a few types of data issues and present real-world examples which illustrate their business impact. Our hope is that reading this paper will help you to appreciate how data quality contributes to software quality, and equip you with a set of tools for capturing better data.

Biography

Danny Casale is a Database Developer and Data Miner at Portland General Electric in Portland, Oregon. He specializes in information systems and data analytics. Danny has a B.S. in Information Technology from Florida State University and M.S. degrees in Management and Information Systems and Operations Management from the University of Florida.

Dan Elbaum works as a Senior IT Business Analyst at Con-Way Enterprise Services where he focuses on mobile applications for truck drivers and freight handlers. He has been specializing in information systems analysis and design since 2007, when he graduated with an M.S. in Management from the University of Florida. He is an IEEE Certified Software Development Professional, and an INCOSE Associate Systems Engineering Professional.

Copies may not be made or distributed for commercial use

1. Introduction

Requirements elicitation has historically been a process of identifying what services an application should provide rather than what data an application should collect. This bias towards functional requirements made sense in the past when storage capacity and network bandwidth were too limited to collect and store large volumes of data cost-effectively. In that historical context, applications only collected the minimum data necessary to fulfill their functional requirements, and enable user-facing features to work correctly.

In recent years, data requirements have grown in importance. We now have the capability to collect large volumes of data and extract insights in near real-time, using more powerful analytical techniques than were previously available.

The purpose of this paper is to help application analysts and developers identify opportunities to collect useful data which may be valuable to store and analyze. We want to help readers think broadly about decisions that need to be made about data that an application handles.

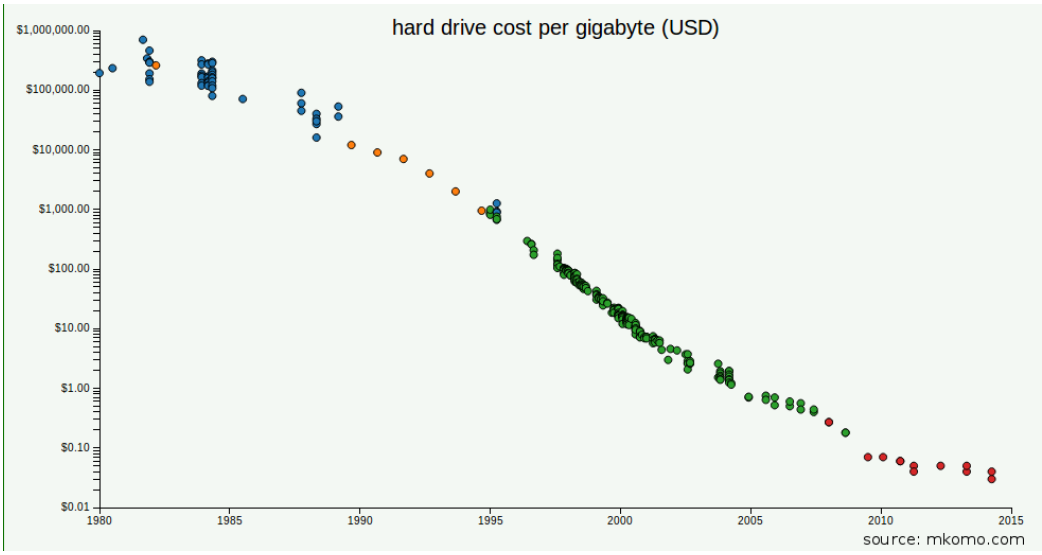
This paper is divided into four sections. The first section reviews recent advances in data storage and analytics which make data more valuable to collect than it was in the past. Section two explains an approach for how to identify what data to collect about each event that occurs within an application. Section three covers how to store the data we've collected. Section four covers how to manage changes to the data after it is already in storage. In each section, we will walk through examples in which we describe a business event, review the possible data that could be captured about it, and discuss the implications of data-related decisions.

2. Advances in Data Collection and Analysis Technology

Storage capacity, processing power, network bandwidth, and data analysis tools have all radically improved in performance and plummeted in price over the course of the last decade. These advancements, which we describe in greater detail below, enable the software development community to capture a larger volume of data, and extract more value from it than ever before.

2.1 Data Storage and Throughput

The cost of data storage has plummeted in terms of dollars per gigabyte ($/GB) over the last 35 years. Note that the vertical axis in the figure below is on a logarithmic scale.



As of August 2014, Amazon offers storage for 1¢/GB per month (Amazon, 2014).

Data storage technology has improved dramatically because better RAID techniques, hardware and caching systems allow for the I/O and throughput necessary to handle massive-scale applications. The commoditization of 10Gb Ethernet means standard networking equipment has the bandwidth to transfer huge amounts of data. As a result of cheaper storage and processing, database size is no longer the primary limiting factor on system availability and performance in most applications.

2.2 Data Analysis and Analytics

The affordability of storage capacity has motivated accelerated development of powerful new tools and techniques for analyzing large datasets. For example, advanced software-enabled analytic techniques like machine learning are gaining popularity, and in many instances partially replacing or automating previously manual statistical analysis techniques. Historically, organizations seeking to predict future events would hire statisticians to apply inferential statistics (analyzing a representative sample and extrapolating to the greater population). This labor-intensive technique required high-quality data, which is rare in most organizations and susceptible to sampling bias. Now, we can supplement traditional statistics with new computational techniques such as machine learning and run analytics on entire populations rather than just representative samples of a population.

Machine learning algorithms have the capability to process billions of rows of data in real-time, continually learn about the anomalies in the data, and recursively improve themselves to account for them. This trend of analyzing entire datasets to overcome data quality issues will continue to rise in popularity as database, storage and computing technology advances.

Taken together, all of these advancements improve the return on investment and time-to-value from data collection. In the next section, we'll introduce some tips for thinking through an application's data requirements so that readers can better acquire the data necessary to harness the power of the new capabilities that we have described in this section.

3. Identifying What Data to Collect

So we're building an information system that persists data, how do we determine what data to collect? As a starting point, we should enumerate all events that occur within the application, and then identify every dimension of each event that we could possibly collect data about.

One method for identifying the dimensions of an event is to conduct a "6Ws" analysis:

Who is involved?

What did they do – To what is it done?

When did it happen?

Where did it take place?

Why did it happen?

Ho**W** did it happen – in what manner?

(Corr, L., & Stagnitto, J., 2012)*

**(Corr and Stagnitto define the 7Ws analysis, this paper only uses 6)*

In the sections that follow below, we will walk the reader through some example situations, and apply the 6Ws analysis to an event in order to show how we explore the issue of what data to collect.

3.1 Example: Customer Payments

Companies usually accept customer payments through multiple channels, such as web, phone, mail, or in-person at a physical office. In one problematic instance, we've observed a system that failed to record which channel each payment was made through.

When the company's physical offices were up for renewal, the company was faced with the decision of whether to renew the lease for the existing offices, or rent in different locations. In order to inform this decision, the company would have liked to explore data on which offices were being used by which customers and how.

Unfortunately, the system didn't collect data about which channel payments were made over, much less which physical office they were made at. The problem here was that the data model didn't include an attribute called something to the effect of "PaymentSource" that would serve to identify the Office, or more generally the channel that the payment was made through.

Though this data element was not necessary to fulfill the functional requirement of processing customer payments, the omission of this data element limited the company's ability to make an informed decision about where to rent their community offices.

Analyzing the Example

Had a 6Ws analysis been conducted on the customer payment event before the system was built, someone may have recognized that the system failed to collect data to answer the 'Where' question.

Below is a reasonable 6Ws analysis of the customer payments event.

- Who: Which customer submitted payment? Which customer service representative serviced the transaction, if any?
- What: What was the dollar amount of the payment?
- When: When did the customer make the payment?
- Where: Where was the payment made? Was it made in a physical office? If it was made online or through a mobile device, can we collect any geo-location information?
- Why: Why did the customer submit a payment? What services was the payment applied towards? Was it applied to specific charges, or to a general account with multiple charges?
- How: How did the customer pay? Did they pay with credit, debit, cash, or e-check? Did they submit payment using a mobile phone? Did they pay the bill all-at-once, or through installments?

3.2 Alerts

Businesses sometimes send "alerts" to notify customers of their upcoming bills, mobile data usage, or electricity consumption. Understanding how alerts influence customer behavior is valuable because it allows companies to continuously improve their customer interactions, and better model and predict customer behavior.

We have seen an implementation of alerts that collected insufficient data for analysis, and precluded continuous improvement of the system. The system at hand stored a user's enrollment status in alerts (subscribed/unsubscribed), but did not maintain a log of all alerts that had been sent. It simply kept a monthly count of how many alerts were sent out to the entire customer population, how many customers subscribed during the month, and how many cancelled their subscription during the month.

This implementation was problematic because a large volume of customers were unsubscribing from alerts, and the business didn't have data to help them decide what approach they could adopt in order to

reduce the attrition rate. They had some hunches that the alerts may've been delivered too frequently, or at not the best times, but they didn't have any data to verify those hypotheses.

Analyzing the Example

Had an analyst conducted the 6Ws analysis on the alert event before building the system, they may have recognized that the system failed to collect data to answer the 'when' question, as well as to collect other data which would've helped the company to develop more robust models of customer behavior.

- Who: Which customer received the alert? What time zone do they live in?
- What: Which alert did the customer receive?
- When: At what date and local time did the customer receive the alert?
- Where: Where was the customer when they received the alert?
- Why: What triggered the alert?
- How: Which channel/device did the customer receive the alert on/through? Was the alert delivered via text message, email, or an automated phone call?

4 Determining How to Store Data

We will now discuss how to store the identified values of each dimension in a way that makes the data more susceptible to analysis. In this section, we focus on the importance of capturing data at the appropriate level of resolution, and the importance of capturing data in a structured format where possible.

4.1 Resolution

Resolution is the granularity of measurement at which a numeric value is stored. Sometimes, categorically new insights can be extracted from granular measurements that are not feasible to obtain from coarse measurements.

4.1.1 Example: Electricity Usage Data

Electrical utilities typically record electricity usage by reading customers' electrical meters and measuring the cumulative number of Kilowatt hours consumed per 15 minute interval. At this resolution, the data does not show how consumption is distributed within each 15 minute interval, nor how electricity usage fluctuates at the sub-kWh resolution. This relatively coarse data resolution was likely adopted before people realized that important insights can be extracted from higher resolution electricity usage data. For example, tools exist that can take as input watt-hour usage data, run analytics, and disaggregate it to determine what types of devices or appliances were likely running, such as heating, cooling, solar, appliances, or malfunctioning equipment. This data can also be used to identify power theft, and forecast load, among other applications.

Analyzing the Example

Had we taken the event "Electrical Meter Read" through a 6Ws analysis, we would have identified that we were storing data at an inadequate resolution to fully answer the 'What' and "When" questions.

Below is a sample analysis of the event "Electrical Meter Read".

- Who: Which customer consumed the electricity?
- What: What was measured? (Ideally this would have been watt-hours.)
- When: When was the meter read? For what time interval did we receive usage data? How frequently should we record measurements?
- Where: Where was the meter read from, remotely or on-site? Where is the meter located in terms of both street address and latitude/longitude coordinates?

- Why: Why was the meter read? Was it read at a regularly defined interval, or for another reason?
- How: How was the meter read, by a person physically present on-site or through remote methods?

4.1.2 Example: Cross-Channel Analytics

Companies often engage with their customers through a variety of channels such as phone, Interactive Voice Response (IVR), web, storefronts and mobile. The channels which do not require human labor to operate are classified as “self-serve”, and cost substantially less to maintain than non-self-serve channels. The cost of self-serve channels such as web and automated phone systems typically cost on the order of a few cents per customer engagement, whereas phone or storefronts can range in cost from $10-30 per customer engagement (Voxeo, 2013). Thus, there is a clear business case to influence customers to engage the business through self-serve channels like mobile, IVR and the web. So how do businesses achieve this goal? One way is to apply cross-channel analytics. In cross-channel analytics, businesses look at the channel that each customer starts in, and tracks the customer to see if they fall out of the self-service channel at some point, and into a higher-cost channel. Machine learning and other data-driven techniques can then be applied to learn why customers are falling out of the low-cost channels. This information can then be applied to help companies correct those issues and attract customers back into the self-serve channels.

One mistake that can prohibit cross-channel analytics is storing the timing of customer interactions at too coarse of a resolution. When timestamps are captured at day-level resolution rather than second-level resolution, it’s not possible to determine the sequence in which same-day events occurred. So suppose we know that a customer has engaged the company both through phone and web on a single day to find information – the company would have no way of knowing whether the customer began their interaction with a call, and then visited the company’s website, or vice-versa.

Analyzing the Example

- Who: Which customer interacted with us?
- What: What did the customer do to interact with us?
- When: When did the customer interact with us? This should be a date/time to enable event-sequencing across channels.
- Where: Where was the customer when they interacted with us? On a particular website screen? Do we want to record the IP address for geospatial analytics?
- Why: What was the customer trying to do? Were they at a particular step in a sequence, like bill payment?
- How: How did the customer interact with us? Did they land on that webpage by directly accessing the page, clicking on a link in an email, or another pathway? What operating system or browser were they using?

4.2 Unstructured Data

This section reviews the importance of collecting data in a structured format rather than through free-form text input fields. When collecting data from users, it can be tempting to take the easy path and allow free-text input instead of doing additional work to figure out a set of pre-determined options for users to select from. While free-from text input fields may be less restrictive for users, and easier to technically implement for developers, they generate data that is more difficult to aggregate and analyze in bulk. The better option from an analytics standpoint is to anticipate what type of feedback to expect from users, and then make users select from one of many exclusive choices so that their feedback can be aggregated and analyzed more easily.

4.2.1 Example: Marketing, Unsubscribe

Consider the data that companies attempt to collect when people unsubscribe from their emailing lists. All promotional emails are legally required under the CAN-SPAM act to provide a means of unsubscribing (FindLaw, 2014). Often, companies use an unsubscription confirmation page to collect customer feedback and learn how to improve their communications. This feedback can be collected with a free-text form, a list of mutually exclusive reasons, a list of non-mutually exclusive reasons, or any combination of the above.

One common error we've seen on unsubscribe pages is that companies ask for free-text commentary about the reason people have unsubscribed.

Analyzing the Example

It is advantageous to ask customers to provide feedback by selecting from a pre-determined list of reasons that map to actions that the company can take to rectify the problem that the customer identifies. For example, if the customer says that the email is too frequent, the business can adjust the frequency of emails or allow the customer to select the desired frequency of emails. If many customers are giving feedback to the effect that content is not relevant, perhaps the business should consider offering segment-specific messaging or more targeted offerings. By offering a list of issues, businesses can help customers to identify correctable issues. On the other hand, offering a free-text field may help capture issues the business did not think of.

As a note, we recognize that the availability of natural language processing software is increasing, but most companies do not have the software and required skills to perform this type of semi-automated text analysis.

5. Handling Changes to Stored Data

Handling changes to already-stored data within an application is another important opportunity for improving the quality of data captured by software. Changes to data include updates, deletes and inserts. There are two main aspects of data changes to deal with: how to archive previous values, and how to capture attributes of the event that changed those values.

With respect to archiving previous values, the safest option from a data retention perspective is to never delete or overwrite any values, but rather archive outdated record or attribute values in a history table. The never-delete, never-overwrite plan is not practical or necessary in most cases, so a middle-ground must be established on a case-by-case basis.

In addition to the archival of previous values, one should also consider recording attributes of the change event itself, such as when the change happened, who made the change, where the change was made from, why the change was made, or what system the change was made through. For example, if a value is updated, consider whether there may be benefit to not only archiving the previous value, but also storing who updated the value, and why the value was updated.

Although data warehousing can solve some of these issues described above, many applications have no data warehouses, and applications are often initially launched before integration into a data warehouse occurs. Additionally, for certain tables that don't change often and/or are small, keeping historical records may be more efficient than implementing an ETL/ELT process which pulls that data every minute or attaches triggers to the table.

5.1 Example: Company Relationships

One scenario where change data can be important is measuring inter-company relationships that span a period of time. For example Businesses often track prospective customers in a database, and those customers are often businesses. Many businesses are interlinked in a family of related business entities,

such as a conglomerate consisting of multiple subsidiaries. Each business entity could be acquired, sold, consolidated or closed. How do we handle these events from a data perspective? Should we store only the current relationship status and the date it came into effect, or the start/end date of the current relationship and all prior relationships with their corresponding dates. Such historical records may not be necessary to fulfill the functional requirements of a sales application, but they may be necessary to produce a report of, for example, how customer payments roll-up into parent companies which may have re-structured over time.

Analyzing the Example

When a change occurs in a parent-subsidiary relationship between companies, what information might be valuable to capture?

Below is a reasonable 6Ws analysis of this scenario.

- Who: Which companies participated in the relationship?
- What: What type of relationship was created between the companies?
- When: When did the relationship begin and end? Do we need just date or date and time. Do we need to allow for back-dating and future-dating for accuracy reasons?
- Where: Where is the relationship valid or where was the relationship formed? Is the relationship only valid between certain countries or regions?
- Why: Why was the relationship formed?
- How: How was the relationship created? For example, was it through a merger or acquisition?

6. Conclusions

Data requirements are an important aspect of software requirements to consider when planning new applications. In thinking about data requirements, we should focus not only on what data must be collected in order to fulfill functional requirements, but also what data might be valuable to analyze in order to extract insights. In thinking through data requirements, we recommend that readers think through each dimension of an event that data could possibly be captured about, how to store that data, and how to capture changes to that data after it is already in storage.

The reader is cautioned, however, that this paper merely outlines a set of tips for thinking through data requirements, and not an exhaustive, holistic approach. There are many other factors worth considering in evaluating how to capture data with software applications.

References

1. Corr, L., & Stagnitto, J. (2012). *Agile data warehouse design: Collaborative dimensional modeling, from whiteboard to star schema* (Revised/Expanded ed., p. 10). Leeds: Decisionone Press.

2. Komorowski, M. (2014, March 9). A history of storage cost (update). Retrieved August 15, 2014, from http://www.mkomo.com/cost-per-gigabyte-update

3. Amazon (2014). Simple Storage Service. Retrieved August 15, 2014, from http://aws.amazon.com/s3/pricing

4. The CAN-SPAM Act: Trying to Protect Consumers From Unsolicited Commercial E-Mail - FindLaw. (n.d.). Retrieved August 16, 2014, from http://consumer.findlaw.com/online-scams/the-can-spam-act-trying-to-protect-consumers-from-unsolicited.html

5. The Power of Personalization: Optimizing Customer Self-Service for Increased Loyalty and Cost Savings. (2013, June 12). Retrieved August 15, 2014, from http://www.voxeo.com/pdf/Personalization_Whitepaper_June12.pdf