Chapter 3

Guerrilla Analytics Principles

3.1 MAINTAIN DATA PROVENANCE DESPITE DISRUPTIONS

The previous chapter's challenges and risks may have been familiar to you. You may have finished that chapter feeling a little depressed by all of these challenges and risks posed by analytics projects. Nonetheless, we cannot escape the fact that analytics work is complex and Guerrilla Analytics projects even more so. There are so many moving parts across the data, the requirements, the understanding of the data, and in the team itself. This is made worse by the constraints on available tools, resourcing, and the requirement for reproducible and traceable work products.

Looking back through the multitude of challenges and risks described in the previous chapter, you may have noticed a common theme.

- There are many disruptions presented to the team. Examples include data refreshes, changing resources, unforeseen data issues, and changing requirements as data understanding grows.
- These disruptions present risks and the effects of these risks are confusion, incorrect analyses, wasted time in repeated communication, inefficiencies, and ultimately project chaos. The team loses track of the latest data and business rules. Code bases are poorly maintained and versioned.

The key insight that drives the Guerrilla Analytics Principles is that the primary cause of these risks is a lack of data provenance. Data provenance is the ability to trace where data came from, how it was changed, and where it ended up. If data provenance could be maintained then a team would be robust against the inevitable disruptions to their work because they would know the following.

- All data and versions ever received by the team and any associated issues affecting that data.
- All versions of business rules derived by the team or dictated by the customer and when those rules came into effect.
- All data manipulations and changes performed on the raw data received by the team to produce work products issued by the team.
- Every work product created by the team as well as when it was created, who worked on it, and who it was delivered to.

- All work products would be reproducible, given the raw data and the particular business rules that were known at the time the work product was created.

The good news is that maintaining data provenance is possible despite the disruptions of a Guerrilla Analytics project. It requires keeping in mind only a small number of guiding rules when doing analytics work. These are the Guerrilla Analytics Principles.

3.2 THE PRINCIPLES

3.2.1 Overview

The Guerrilla Analytics Principles are as follows.

- **Principle 1:** Space is cheap, confusion is expensive.
- **Principle 2:** Prefer simple, visual project structures over heavily documented and project-specific rules.
- **Principle 3:** Prefer automation with program code over manual graphical methods.
- **Principle 4:** Maintain a link between data on the file system, data in the analytics environment, and data in work products.
- **Principle 5:** Version control changes to data and program code.
- **Principle 6:** Consolidate team knowledge in version-controlled builds.
- **Principle 7:** Prefer analytics code that runs from start to finish.

The Guerrilla Analytics Principles are not prescriptive. As discussed earlier, data analytics is a wide-ranging field that encompasses many activities and technologies. Instead, the principles aim to help you make the right decisions in coordinating your analytics work so that data provenance is preserved. The resulting benefit is that the challenges and risks of the Guerrilla Analytics project can be addressed.

As we elaborate on these principles throughout the book, you will see that they borrow from fields such as Agile, Extreme Programming (Kent Beck, 1999), Test-Driven Development (Beck, 2002), Continuous Integration (Duvall et al., 2007), and Continuous Delivery (Humble and Farley, 2011). A familiarity with these topics would certainly help but is not necessary in understanding and implementing the Guerrilla Analytics Principles. The subsequent sections now explain the principles in more detail.

3.2.2 Principle 1: Space is Cheap, Confusion is Expensive

Data takes up storage space and storage space costs money. While this consideration is important, especially with large data volumes, it is surprising how teams will scrimp on trivial amounts of storage space.

Teams can lose hours of project time because they failed to retain a back-up copy of data on the file system, when data had been lost from the Data Manipulation Environment. Similarly, I have seen hours lost in tracing the source of a work product because an analyst jumped straight into modifying a spreadsheet from a customer without taking a copy of the original file. Often, as work product versions evolve, the older versions are overwritten and replaced instead of archived. This causes confusion if multiple versions are "in the wild" with the customer and cannot now be reproduced.

This principle emphasizes that when faced with space considerations, always think about the impact on loss of data provenance. Ultimately, this is more expensive for the team than trivial amounts of storage space.

3.2.3 Principle 2: Prefer Simple, Visual Project Structures Over Heavily Documented and Project-specific Rules

If project folder structures are complex, then busy data analysts will not have time to understand and follow them. If database structures are complex, then new team members struggle to roll onto the project and agility is compromised. Not only do team members struggle to "find things," they become overwhelmed when deciding where to "put things." This problem is exacerbated when your teams move between several projects and find that each project has its own specific rules and conventions.

The more visual and simple a project structure is, the easier it is to understand and follow. This principle applies to all project folders, data environment structures, and team conventions.

For example, consider the project structures shown in Figure 8. In the first folder structure on the left, it is impossible to know where to store data. The data folder seems to be categorized by type of data and then by receipt date. Analytics work is scattered in a variety of folders, again with a nonsense categorization by analytics programming language. The project has been broken up into many components and some of them seem to overlap. The result is that rather than incorrectly storing an analysis, data or documentation, the teams instead create their own structures that each of them understands individually. Over time, the project's structures deteriorate to the point that nothing can be easily found.

In the second scenario on the right, the core functions of the project have been identified and assigned a single folder. All data goes in one place. All analyses go in one place. You do not have to think about where to locate things because it is obvious. Each team member has his/her own workspace for everything temporary he/she may need to do.

You could think of this principle as "convention over configuration." The approach is straightforward and the simple naming convention is understood by looking at the project folder rather than remembering some complex project

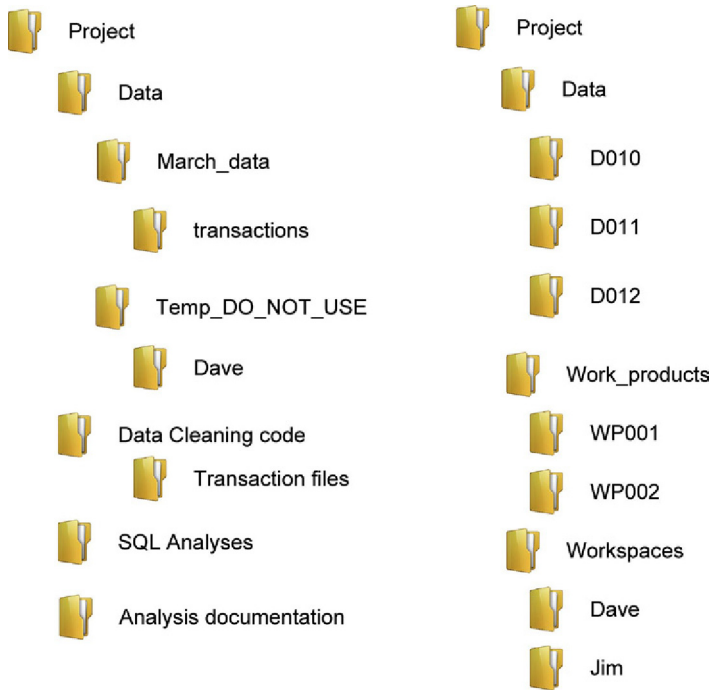

FIGURE 8 A complex project structure and a simple, visual project structure

configuration guideline. Similar conventions should be applied in the Data Manipulation Environment.

3.2.4 Principle 3: Prefer Automation with Program Code Over Manual Graphical Methods

Program code is arguably the most expressive tool we have for clearly and definitively describing our analytics work. Program code explicitly describes how data is being changed, the rules that are being applied, and the correctness checks that are done.

However, there are also analytics tools that allow you to work through a graphical user interface rather than use program code to manipulate data. Use of these tools has implications for data provenance.

Think about data manipulated in a spreadsheet. A typical workflow could involve copying data to a new worksheet, dragging a formula down a column to apply it to some cells, and then pivoting the final worksheet. These manual operations of copying and dragging are not captured anywhere in the spreadsheet and so are difficult to reproduce.

This principle states that as far as possible, analytics work should be done using program code.

3.2.5 Principle 4: Maintain a Link Between Data on the File System, in the Analytics Environment, and in Work Products

When aiming to preserve data provenance, it is helpful to think of data flowing through your team and your analytics environment. Imagine a data point arrives with your team (perhaps accompanied by several million other data points). The data is stored on a file system before being imported into the Data Manipulation Environment. The data is then manipulated in several ways such as cleaning, transformation, and combination with other data points. The data then leaves the team as one of the team's work products. At each of these touch points, the data is potentially being changed by one or more team members and one or more processes. Problems arise when a work product's data is incorrect or is questioned by the customer. Errors could have been introduced at any stage of the data's journey from raw to work product.

This principle emphasizes that you should strive to maintain the traceability of the data on this journey. You should be able to look at a data point in an output work product and quickly trace its origins back to its raw source, if necessary.

3.2.6 Principle 5: Version Control Changes to Data and Program Code

If Guerrilla Analytics is essentially about preserving data provenance in spite of disruptions, then we must consider the factors that affect data provenance. The provenance of analytics work is broken if any of the following change.

- **Raw data:** If raw data changes then all analyses built on top of that raw data may also change. A raw data change could consist of renaming a dataset, changing a dataset location, or changing the contents of the dataset in terms of either its fields or field values.
- **Code that manipulates the data:** If you modify your analytics code then your analyses will change. These modifications could arise because of a change in business logic, addition of cleaning rules or any seemingly innocent modification of analytics program code.
- **Common routines applied to the data:** Your analysis code may call out to common routines built by the team such as cleaning routines. If those routines change then your analysis will also potentially change.
- **External software and libraries used by the code:** Your analysis may use a third-party library or a particular version of a programming language. If those libraries and languages are updated or changed by the team then potentially your analysis outputs will also change.
- **Tuning parameters:** Many analyses, particularly statistical analyses and machine-learning algorithms, have a variety of tuning parameters that change their behavior. In a clustering problem, for example, you may specify the number of clusters or threshold distances between clusters as an input to the algorithm. In a statistical model, you may specify a significance or

power level for statistical tests. Changing these types of input parameters will change the outputs of the process.

This principle emphasizes that you should control changes in any of these factors. You can do this by putting in places a few simple checks and rules.

- Never modify raw data after it has been imported into the Data Manipulation Environment.
- Once analytics code has been used for a work product, never modify that code. When the code needs to be revisited, make a new version of the code.
- Be clear on the versions of common routines used by your code and make sure that those common routines are themselves version controlled.
- Be clear on the versions of program languages and libraries used by your code and never replace those languages and libraries. If languages and libraries need to be upgraded then do so while also keeping the older versions available for reproducing existing analytics work products.
- Record input parameters to algorithms and statistical tests so that they can be rerun and produce the same outputs.

Subsequent chapters will elaborate on this principle throughout the analytics workflow.

3.2.7 Consolidate Team Knowledge in Version-controlled Builds

As the project progresses, understanding of the data will grow and evolve. For example, team members may learn that a particular data source needs to be deduplicated before it can be used. The customer may specify a certain business rule that they would like to see in all work products. A coded field value such as "MPR100" might have an agreed business translation. These pieces of team knowledge must be centralized somewhere in the team for consistency and efficiency.

An analytics build is a centralized and version controlled data structure or analytics functionality that captures team knowledge and version controls it.

Consistency is achieved because data knowledge now resides in one location. Efficiency is achieved because much of the hard work of preparing data and solving technical problems has already been done and is published to the entire team in the build.

Builds are version controlled because team knowledge grows and changes during the project. Older work products were built on older knowledge represented in older versions of the build.

3.2.8 Principle 7: Prefer Analytics Code that Runs from Start to Finish

Analytics code by its nature is exploratory. A data analyst will summarize and profile data and apply many transformations to it as they seek to understand the data. Creating an analytics model is an iterative process of creating new

variables, scaling them, and testing whether they are good predictors for a variety of model types, such as regression or decision trees. If not managed carefully, this exploratory nature of the work can lead to poor coding practices. Code snippets to profile data and test hypotheses linger in final work product code. These snippets clutter program code files with code that is not needed to reproduce the work product. They very often break code execution with the result that reproducing the work product necessitates a time-consuming stepping over the older code snippets. Code reviews are more time consuming than necessary because a reviewer must work out which snippets to avoid. Since code is cumbersome to execute, execution is delayed or avoided and this increases the risk of bugs and broken data provenance going undetected.

A simple principle of delivering code that executes from start to finish eliminates these problems while still accommodating the need for data exploration and iterative model development.

3.3 APPLYING THE PRINCIPLES

Guerrilla Analytics projects can be successfully managed and the risks of the project environment can be mitigated by following the Guerrilla Analytics Principles. The question then becomes, how to implement these principles and how to manage a Guerrilla Analytics team.

First of all, let us acknowledge that a top-down management approach to Guerrilla Analytics projects will not work. Analytics projects are complicated and Guerrilla Analytics projects are arguably more so. The natural human reaction is to "manage" this complexity through rules and conventions. This is a top-down approach. Some typical symptoms of a top-down management approach and their impact include the following.

- **Controlling complexity with project folders:** A complex project folder structure that is for the convenience of management instead of the reality of data analytics work. Management attempts to "standardize" an analytics project structure with categorizations that often look like the following.
  - Requiring that emails be stored in a "communications" location separate from the work they are related to. The intention is to be able to easily identify all communications with the customer. The effect is that links have to be created between communications and their associated data and work products in another project location.
  - Requiring that stages in the analytics workflow such as data cleaning have a distinct tree of folders separate from other stages in the data analytics workflow. The effect is that it becomes very difficult to group together all code files associated with a particular work product.
  - Deciding that folders should be categorized by programming language so you have folders such as "SQL code." Presumably Python code, Java code, spreadsheet analyses, Tableau dashboards, and all other data manipulation languages get their own project folders too?

- **Controlling delivery with process:** Imagine a cumbersome process that analysts must follow to create and deliver their work. In one instance, to improve traceability of work products, the team was instructed to copy all delivered work products into a deliverables folder. Needless to say that under pressure, quick fixes were made to work product copies in this deliverables folder rather than the original project location. Deliverables went out of sync with their original code and data. The impact here was confusion and inconsistency in delivered work products.
- **Pushing administration down to analysts:** Analysts are busy summarizing their work for project managers because only the analysts can keep pace with the details of the high volume of outputs being released to the customer. Detailed data dictionaries, data logs, and requirements documents are requested in an attempt to maintain oversight of what the team is doing. These documents cannot be produced and maintained at the pace of the project's delivery and evolving understanding and so the documentation goes out of date and is not useful.

You may have many examples of your own. Perhaps as a manager you are guilty of inflicting top-down control on your team so that you and your stakeholders have more comfort in your team's outputs. This approach simply doesn't work. It takes analysts away from the activities where they can add most value. The manager and project manager are often little wiser in their understanding of their team's outputs. The processes themselves are complicated and amount to little more than a box-ticking exercise. This approach does not scale and the team simply will not follow the top-down process in a highly dynamic Guerrilla Analytics project.

I have thought a lot about these challenges over the years and tried several approaches to managing Guerrilla Analytics projects with the input of the teams I have managed. A successful Guerrilla Analytics management approach should have the following characteristics.

- **Bottom up:** There should be a structure and order to the project that emerges from the work of the analysts and complements the work of the analysts rather than being imposed to suit a project management agenda. If done correctly, this bottom-up approach can also satisfy project management requirements.
- **Not disruptive:** As far as possible, the analysts should be able to satisfy the requirements of coordinating the project and reducing the analytics risks without having to interrupt their workflow for documentation, box checking, or some other burdensome administration.
- **Simple and lightweight:** There should be minimal overhead of documentation, rules, and conventions that the team needs to absorb and remember.
- **Consistent:** As far as possible, all data should be treated in a similar fashion, all code should follow some basic conventions, and all work products should have a similar structure. This minimizes what the team needs to learn and so keeps the team flexible and agile.

The subsequent chapters will describe how to implement Guerrilla Analytics Principles throughout the analytics workflow. As you go through the workflow stages, note how these four management characteristics are always present.

3.4 WRAP UP

This chapter has introduced and described the Guerrilla Analytics Principles. You should now be familiar with the following topics.

- **Maintaining data provenance despite disruptions:** What makes Guerrilla Analytics projects particularly challenging is the variety and frequency of disruptions. Data is changing, business rules are evolving, resource is changing, and requirements are changing. This causes data provenance to break. The team loses track of where data is located, how that data was changed, and where analyses based on that data were delivered. If data provenance can be maintained, many of the risks and challenges of an analytics project with frequent disruptions are mitigated and overcome.
- **The Guerrilla Analytics Principles:** These are a small set of lightweight rules of thumb that promote data provenance and so help in the management of the risks and challenges of a Guerrilla Analytics project.
- **Managing Guerrilla Analytics projects:** To apply the Guerrilla Analytics Principles successfully, a management approach must be adopted, that is:
  - Bottom up
  - Simple and lightweight
  - Not disruptive
  - Consistent