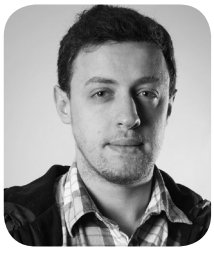
Adapting Design Thinking to Agile Scrum DW/BI Development

Pedro Cerqueira and João Brandão

Pedro Cerqueira is a DW/BI product owner at Farfetch.
apc1986@hotmail.com



João Brandão is a data and artificial intelligence engineer at Bosch.
jpabrandao@gmail.com

ABSTRACT

Many enterprises now see data warehouse/business intelligence (DW/BI) solutions as primary strategic assets. At the same time, agile development methodologies such as scrum are becoming more popular in DW/BI, mainly because they bring a completely new mindset to the development process. Primary benefits are related to the fast pace and efficiency that DW/BI developers can achieve when working as a team. However, as DW/BI developers grow accustomed to relatively new concepts and ways of working, they should be aware of problems that may arise from flying so fast and at such low altitude.

This article describes how to adapt the design-thinking approach to solve problems in agile scrum DW/BI development, which we will illustrate by implementing a user story from start to finish. The aim of this new approach is to help DW/BI developers work with a creative mindset and more as a team by iteratively improving the quality of the process and product in an agile context. We also show how the approach can help DW/BI developers change, tailor, and improve their thinking about the traditional DW/BI development process.

INTRODUCTION

Usually, when picking a new DW/BI story from their scrum board, most developers just pick any story, try to understand it as best as they can, and then envision and design a way to implement it. After all the implementation work is done, they will think of several

ways to test it, and then pass it on to code review and testing.

As simple as this seems, anyone in the DW/BI industry knows that the development process entails a huge number of specific data and performance challenges, which often result in a very complex development task. However, what if a DW/BI developer had a list of specific steps he or she could consistently use that would significantly lower the chances of failure?

Using design thinking, developers are encouraged to be creative and are excited to think in new ways about DW/BI development solutions. Using this approach, developers feel more confident that their solution is the best one possible both in technical terms and for stakeholders' use. This approach also encourages sharing problems and solutions, which fosters the emergence of different points of views and a continuous improvement mindset.

In the next sections, we will present an approach based on design thinking that has been specifically tailored to DW/BI development and will illustrate how to implement it with an example user story.

UNDERSTAND

The first step is to fully understand the user story. As part of this process, the DW/BI developer should diligently read the story card and any comments written on it. As you read, try to remember the conversation that took place during the refinement meeting when this particular task was groomed. Also, any acceptance criteria or tests that might have been written on the card could be of great value and should not be dismissed. This is known as the three *C*'s technique—*card, conversation*, and *confirmation*.

If doubts or questions crop up, swiftly get feedback from the product owner, tester, or other developers. Rely on others who are more experienced in a given area or who might have kept more information from the refinement session conversation. Quick feedback can save precious development time and helps clarify ideas and assumptions. This is also where you should try to empathize with the final users to take all their possible needs and pains into consideration. Try to focus on the expected outcome.

Quick feedback can save precious development time and helps clarify ideas and assumptions.

After choosing a story, begin by getting clarification on the overall purpose of it and understanding the operational source system and data that will be used as the basis for the development task. This is integral to the decisions you may have to make during the definition and exploration phase. Beware of making assumptions about existing operational sources and try to consult sources such as source code, documentation, data models, architects, and subject matter experts (SMEs).

As an example, imagine an experienced DW/BI developer named Ana working in a scrum team in a global e-commerce company. Her team is committed and she is now ready to choose her first story for the upcoming sprint. After considering the story's priorities and her skills, Ana and her team jointly decide that she will work on the following user story:

AS A data analyst

I WANT TO analyze data containing information on the customer's billing address in the sales data mart

SO THAT I can understand data patterns and create reports regarding customer orders in which the billing address is different from shipping.

At first glance, it might seem that the request is simply to build a dimension containing billing information and cross it with orders information. However, Ana should not assume that the billing source information has the minimum required quality or is always available with the right timing for extraction and use in a data warehouse. In addition, many companies still do not have perfect alignment between operational and analytics strategies, meaning that some existing operational features might not be ready for analytics purposes. Therefore, she should find out more about the source data before attempting any major implementation.

Imagine, for example, a situation in which a mandatory field for the customer's billing city exists as part of the checkout process. This particular field is handled as free text in an operational form the customer fills in during purchase. Sometimes the customer types in the billing city's full name (e.g., "New York City") and other times just the initials (e.g., "NYC"). Because the company's operations and transportation services are handled manually and the volume of orders is still relatively low, this is not a problem for them, however it is not sufficient to guarantee adequate data quality for BI purposes.

Nevertheless, if Ana assumes that this particular field has the required data quality without checking, she may be surprised at the final stage of ETL development and reporting to discover that it is actually impossible to return fit demographic data about the billing city—or it may not even be discovered before the product is released for analyst use.

DW/BI developers should watch for data source quality issues by paying attention to themes such as source database referential integrity and data modeling. Another way to minimize risk while being as efficient as possible is to rely on data profiling techniques in order to rapidly spot anomalies that might undermine DW/BI success.

Above all, the developer must apply critical thinking and domain expertise to understand source data and how it can and cannot be used.

You may also want to consider these topics at this stage.

Change data capture If you intend to extract data incrementally, you will need to understand how data changes in the source table. You will need to check if you can use a log table or ID to load only new or updated records. Find out whether the column is being updated and if it is, determine whether to keep the column's historical values in the DW for analytical purposes.

Inferred members If you are considering creating a new analysis axis in an existing fact table, you can achieve this by creating a process to load the new dimension data and then adding it as an analysis axis of a fact table through the insertion of a new surrogate key. With this approach, you need to consider the possibility of encountering source synchronization problems, in which fact

data is coming before the dimension data, which will require the creation of inferred members to avoid losing the early-arriving fact data.

Null Percentages and Distinct Values You may find problems in specific source fields, such as a high percentage of null values. Even if it postpones the story's development, it should be reported to the team as it may mean that you are using the wrong table or data source.

Aggregated Tables When changing a given field, watch for DW aggregate tables. You might unintentionally change the way a field is calculated, which will demand data reprocessing.

Parallel Development This occurs when your development team is working at the same time that the source system and data are being changed. In these situations, you will need to rely on mock or test data to develop the analytics product until the source product is complete, which will limit your understanding of the real problem being solved. Maintain constant contact with the source development software team to ensure alignment between the two projects.

Historical Data One of the most complex problems arises when you need to load historical data that has direct impact on current analysis. Sometimes, when reprocessing historical data, you may find problems that existed in the past and you may need to alter your approach to correct them. Other times, creating updates to correct historical data in addition to current data might become very cumbersome and a cost-benefit analysis might be in order.

For example, in Ana's situation, she needed to add a new dimensional axis. Imagine now that this axis has at least one slowly changing data (SCD) type 2 column—a change management type that inserts new records to track changes—and that the data source does not keep any historical data. As a result, Ana's product will only be able to keep historical data from its release date. This would be obvious to developers but not necessarily to business users, so they should be made aware of this as part of the development process.

In general, it is beneficial to grow your conceptual and technical knowledge about the business and the way it operates. This will be invaluable when implementing a DW/BI solution to ensure that it reflects the way the business functions.

We have also found that having specific DW/BI challenges in mind is even more important when developing in an agile context, as the fast development pace adds to the possibility of losing sight of the bigger picture and may allow smaller user stories to impose on well-intentioned DW/BI developers and throw them off track.

IDEATE, EXPLORE, AND DEFINE

Once the most crucial information about the problem has been gathered, it is time to start thinking of ways to actually resolve it, creatively and with an open mind. There is more than one way of looking at a problem, and the developer should aim to find opportunities for novel or alternative methods. This is where divergent thinking is most valuable.

A good rule of thumb is to define and design at least two distinct solutions, sketch them out in your notebook, and try to identify which one is better in relation to the problem. In doing that, consider both stakeholder and architectural needs. Try to be creative and experiment with different designs and new ways of implementing

them. This is often one of the most enticing parts of being a developer.

If Ana has decided—after the conclusions from the understanding phase in our example—that the best approach is to add a new dimensional analysis axis (billing address) to the orders facts table, she should still consider different options. Simply dealing with basic architecture presents several design decisions.

Here are two approaches she might take and some questions she should ask about each one:

- **Approach 1:** Create an all-new physical billing dimension table.

  If Ana decides to create a new dimension table, she should first find out the likelihood of a huge number of rows finding their way into the table while having just a few columns of SCD type 2 with a high update rate. She may also want to investigate creating a new table dimension containing only affected columns (Kimball's minidimension) in order to avoid concerns with unaffected columns.

- **Approach 2:** Merge the data for the new billing address dimension with the existing dimension of shipping address, resulting in just one address dimension containing all distinct values for billing and shipping.

  In this solution, Ana will want to ask whether she might just refactor existing code to make it better. This would probably increase testing complexity but may provide offsetting savings elsewhere. She may also consider adapting the new billing address to the old shipping address by repeating legacy code.

Both approaches have repercussions for the existing orders fact table: the first would require creating a new surrogate key and updating the history, and the second would require updating existing surrogate keys and respective histories in any fact tables using them.

When considering your own alternative solutions, think about code-related factors and try to keep these best practices in mind:

- Maintain business logic on functions or store procedures so that the logic can be reused by other processes
- Build the ETL in a way that it can be restarted at any time to facilitate the support team's work
- Use consistent nomenclature for better comprehension
- Consider building and using indexes for daily and historical ETL loads, as well as for future business analysis
- Build automated unit tests for better code coverage and safer releases

Additionally, consider exploring new technologies or novel features, such as imagining a new, more efficient type of index for text fields such as billing city.

With all these decisions to make, you are in a good position to find more than one feasible solution. Remember that at this time you are not yet developing anything. You are just pondering different, creative alternatives.

ANALYZE AND DECIDE

Once you have gathered and designed the options, find someone to share, discuss, and

brainstorm with. In an effective team, you should be able to go to another developer without fear of judgment. Reaching out to someone with a different mindset and unique skills—such as a tester or product owner—could prove invaluable. After these quick brainstorming sessions, try to assemble and understand all the input you have received.

There are several likely outcomes from doing this:

- You may find a better solution and decide to try it.
- You may find a better solution but not be able to try it—perhaps because it would take longer than a sprint without justifiable reason or because your organization doesn't have the necessary technology or infrastructure. In this case, you can carry the knowledge of the better solution forward until the right opportunity arises.
- You may even find that your solution is the best solution—maybe as a result of following this approach in the past—and now have shared new information with another developer. You, the team, and the product quality all improve. Everybody wins.

You should now be in a position to make a decision. The decision phase is the part where convergent thinking is most expected.

Returning to our example scenario, imagine that Ana decided to share her two possible solutions (create a new physical dimension table or merge with the existing dimension table) with Maria, a more seasoned developer. Having all the facts in consideration, Maria favors long-term success and advises Ana to implement the merge solution first, as long as Ana and the product owner think it can be done in a way for it to be useful and valuable for the data analysts.

In an effective team, you should be able to go to another developer without fear of judgment.

Otherwise, Maria advises to take a shortcut by choosing the new dimension solution, with the compromise of merging the two address dimensions in the future.

After considering all this and sharing her new insights with the team to avoid unforeseen consequences, Ana can make her final decision.

IMPLEMENT

At this point, there is not much additional design thinking to do. This is where practical technological skills come in. Just grab the final sketch from the preceding step and start developing and prototyping.

Sometimes through rapid prototyping and by involving final users or colleagues when possible, you may discover that your chosen solution is either not feasible or not usable. This can be due to a serious technical or scheduling issue or because you or your team missed some important detail during earlier stages.

Fortunately, because the process is an iterative one in which you should always be inspecting, learning, and adapting, you can go back to the drawing board and select the next most suitable solution from your list of alternatives.

For example, suppose that after deciding to implement the merge solution, Ana discovered that the existing code to load the shipping address dimension was of very poor quality and estimated that it would lead to an additional week of work to fix. Due to the particular urgency of this user story, she decided to follow the advice of the seasoned developer by immediately starting the development of the new billing address dimension. In addition, she promptly created a new story for the future refactoring and merging of both address dimensions as a commitment between the product owner and development team.

TEST

For the purposes of this article, we have separated implementation from testing, but these should occur simultaneously to achieve the best possible outcome. When we refer to testing, we mean all the activities and tasks performed by the developer to reduce risk and uncertainty related to development, as well as anything that helps make sure the finished product responds to users' needs. The focus here is on making sure you are building the right product in the right way. Quality should be always in the back of your mind.

However, this does not do away with a formal testing phase that should be part of the scrum process. Involve the team's tester early in the process to draw important insights from his or her specific mindset and be aware of author bias, which often leads to confirmation bias. Additionally, as you implement and test your solution, you should be thinking about the final users and involving them whenever it makes sense to do so in order to shorten the learning curve.

One of the simplest testing techniques involves comparing row counts between source and destination tables in order to make sure there are no missing or extra records in the target. This technique gives a fast, broad view over the quality of a given process in terms of its integrity. However, this does not guarantee data completeness because even when the number of rows is the same, the content in the target could be different from the source.

As you implement and test your solution, you should be thinking about the final users.

If we want to assure data completeness, we need to also look at possible data mismatches by comparing the content of source tables against target tables. When doing this kind of comparison, be on the lookout for false positives caused by mismatches between ETL auditing and control columns (e.g., row activation dates) that might exist in the DW but not in the source.

Assuming Ana decided to build a new dimension billing process, she could start by comparing source to target after the first data load in order to check whether the number and content of rows is exactly the same between the two.

Another critical issue when dealing with ETL processes is to make sure there is consistency in the several data repositories and staging tables between the source and the final DW tables and views, or even cubes and reports. Column

value and format should always be thoroughly inspected. For example, data warehouses typically lack database-forced referential integrity for insertion-performance reasons (meaning some DWs lack primary keys and not-null constraints). The ETL is usually responsible for maintaining process integrity, but null values and duplicates tend to find their way into some fact/dimension tables. These usually become great candidates for test cases.

As a main part of ETL testing, focus on finding referential integrity problems such as orphan fact records (e.g., a surrogate key in a fact table without a corresponding surrogate key in the dimension table). As an example, Ana could devise a test to find out if any existing orders are missing billing information when joined with the new billing dimension. If the test fails, it could mean that the data lookup was poorly implemented.

Domain integrity is also a major reason for DW/BI failure, mainly due to problems in ETL processes. Test data cases should focus on valid and invalid values in terms of data type, length, defaults, and other rules imposed by database constraints. A good test example would be to have the new billing dimension extract information for international addresses and failing to test for Unicode strings. An error here could result in storing unintelligible data in the DW.

Another typical point of failure concerns the huge number of joins used through the ETL to make sure that starlike fact and dimension tables are as denormalized as they can be in order to provide the high-level performance experience that DW/BI users demand. Always pay special attention to the type of SQL joins in use to understand whether you might be losing rows or adding unwanted ones in the process, as well as to make sure that table grain is always maintained. Investing some time in understanding the technical implementation of joins in the ETL language can have a huge payoff.

Don't forget to test nonfunctional requirements such as performance and reliability.

As a simple example, an inner join with a lookup table that contains duplicates might result in unwanted records being added to the process. When addressing logical duplicates, avoid trusting blindly in artificial keys or even in operational IDs. Rather, try to find the true logical business key for each given table. In the example situation, Ana should focus on finding out if the billing dimension has any duplicate rows. If not handled correctly, this could mean that different surrogate keys in the fact table would be pointing to identical members in the dimension table.

One more testing tip: don't forget to test nonfunctional requirements such as performance and reliability. Restartability is particularly important in the context of DW/BI because of the need to manage and process huge volumes of data—typically in a very short time frame—in order to provide business value.

Last but not least, a DW/BI developer should be aware of data security and privacy problems that may damage a company's reputation or even result in legal action.

After testing is completed, you should be more than ready to confidently deliver the user story for code review and final quality assurance.

CONCLUSION

DW/BI development is a complex art and developers have it rough trying to build processes to handle huge loads of data. In order to smooth the process and increase the probability of success, it is important to follow a systematic iterative process, such as design thinking, that allows you to do the job right while benefiting from the collective expertise of the team.

At the end of this process—after inspecting, designing, learning, adapting, and implementing—you should be confident you have produced the best solution possible and be much more comfortable delivering the product to final acceptance testing and consequent release. ●