Modern Software Engineering Methodologies Meet Data Warehouse Design: 4WD

Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia

DEIS, University of Bologna, Italy

Abstract. Data warehouse systems are characterized by a long and expensive development process that hardly meets the ambitious requirements of today's market. This suggests that some further investigation on the methodological issues related to data warehouse design is necessary, aimed at improving the development process from different points of view. In this paper we analyze the potential advantages arising from the application of modern software engineering methodologies to a data warehouse project and we propose 4WD, a design methodology that couples the main principles emerging from these methodologies to the peculiarities of data warehouse projects. The principles underlying 4WD are risk-based iteration, evolutionary and incremental prototyping, user involvement, component reuse, formal and light documentation, and automated schema transformation.

Keywords: Data warehouse; Design methodologies; Agile development.

1 Introduction

The continuous market evolution and the increasing competition among companies solicit organizations to improve their ability to foresee customer demand and create new business opportunities. In this direction, over the last decade, data warehouses have become an essential element for strategic analyses. However, data warehouse systems are characterized by a long and expensive development process that hardly meets the ambitious requirements of today's market. This is one of the main causes behind the low penetration of data warehouse systems in small-medium firms, and even behind the failure of whole projects [20].

As a matter of fact, data warehouse projects often leave both customers and developers dissatisfied. The main reasons for low customers' satisfaction are the long delay in delivering a working system and the large number of missing or inadequate (functional and non-functional) requirements. As to developers, they complain that —mainly due to uncertain requirements— it is overly difficult to accurately predict the resources to be allocated to data warehouse projects, which leads to gross errors in estimating design times and costs. In the light of the above, we believe that the methodological issues related to data warehouse design deserve some further investigation aimed at improving the development process from different points of view, such as efficiency and predictability.

A. Cuzzocrea and U. Dayal (Eds.): DaWaK 2011, LNCS 6862, pp. 66–79, 2011.

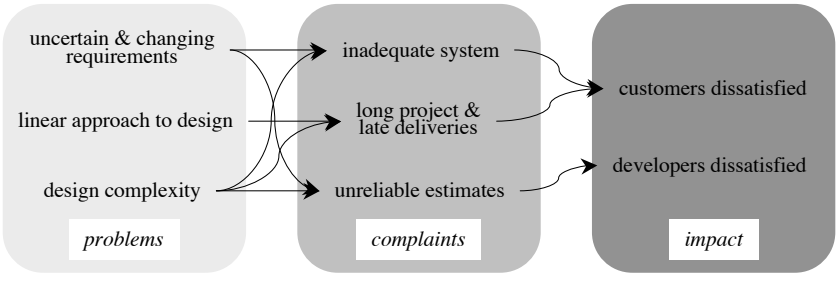

Fig. 1. Cause-effect relationships in customer and developer dissatisfaction

The available literature on data warehouse design mainly focuses on traditional, linear approaches such as the *waterfall approach*, and it appears to be only loosely related to the sophisticated design methodologies that have been emerging in the software engineering community. Though some works about agile data warehousing have appeared [12], there are also evidences that applying an agile approach *tout court* to data warehouse design has several risks, such as that of inappropriately narrowing the data warehouse scope [2]. In this paper we analyze the potential advantages arising from the application of modern software engineering methodologies to a data warehouse project and we propose *Four-Wheel-Drive* (4WD), a design methodology that aims at coupling the main principles emerging from these methodologies to the peculiarities of data warehouse projects.

Our *modus operandi* for this work is the following. First we identify the main problems behind data warehouse projects based on traditional methodologies, and we define our goals accordingly in terms of desired qualities of the software development process (Section 2). Then, from an analysis of the main software engineering methodologies we derive a set of design principles to be adopted in order to achieve the quality goals (Section 3). Then we apply these principles to build up our methodological proposal, inspired by practical evidences emerged during real data warehouse projects (Section 4). Section 5 completes the paper by discussing our proposal in the light of the related works.

2 From Problems to Goals

Our experience with real projects led us to attempt a classification of the main reasons why customers (meant as both sponsors and users) and developers often end up with being dissatisfied. Figure 1 summarizes the results of this investigation, distinguishing between problems, complaints, and their human impact, and emphasizing the existing cause-effect relationships between them. A closer glance at the *problems* column reveals that:

– Requirements for data analyses are often unclear and uncertain, mainly because decision processes are flexibly structured and poorly shared across

large organizations, but also because of a difficult communication between users and analysts. Besides, the fast evolution of the business conditions may cause requirements to drastically change even in the short-term [5]. Failing to address these problems dramatically contributes to making users perceive the system as inadequate from the functional point of view and leads to inflating the overall project duration and cost by introducing unexpected delays in the development process.
- Data warehouses are normally built one data mart at a time; each data mart is developed following a linear approach, which means that the different phases are organized into a rigid sequence. Releasing a data mart requires 4-6 months, and it is very difficult to provide intermediate deliveries to be discussed and validated with users, who may easily feel not sufficiently involved and understood, and loose interest in the project.
- The intrinsic complexity of data warehouse design depends on several issues. Among the most influential ones, we mention a couple: data warehouse design leans on data integration, that in most cases is a hard problem; the huge data volume and the workload unpredictability make performance optimization hard. Problems related to data quality and performances have a particularly negative impact on the perceived system inadequacy.

We argue that these problems can be solved by working on four qualities of the software development process [4], as explained below.

1. The *reliability* of a development process is the probability that the delivered system completely and accurately meets user requirements. In our context, increasing the reliability of the design process can contribute to addressing the "inadequate system" complaint, i.e., to ensuring a high-quality and satisfactory final system.
2. By *robustness* we mean the process flexibility, i.e., its capability of quickly and smoothly reacting to unanticipated changes in the environment. A robust process can more effectively accommodate both uncertain and changing requirements.
3. The process *productivity* measures how efficiently it uses the resources assigned to the project to speed up system delivery. Increasing productivity leads to shorter and cheaper projects.
4. The *timeliness* of a process is related to how accurately the times and costs for development can be predicted and respected. A timely process makes resource estimates more reliable.

3 From Goals to Principles

To understand how the main software engineering methodologies devised in the last thirty years can help designers achieve our four quality goals, we analyzed the objectives and underlying principles of seven methodologies, namely *Waterfall* [21], *Rapid Application Development* [15], *Prototyping-Oriented Software Development* [18], *Spiral Software Development* [3], *Model-Driven Architecture*

[13], *Component-Based Software Engineering* [11], and *Agile Software Development* [1]. Overall, the emerging methodological principles can be condensed as follows:

- *Incrementality and risk-based iteration.* Developing and releasing the system in increments leads to a better management of the project risks, thanks to a proper prioritization of activities aimed at letting the most critical requirement features drive the design of the skeleton architecture. A stepwise refinement based on short iterations increases the quality of projects by supporting rapid feedback and quick deliveries [3,15].
- *Prototyping.* Complex projects are conveniently split into smaller units or increments corresponding to sub-problems that can be more easily solved and released to users. To facilitate requirement validation and obtain better results, system development is achieved by refining and expanding an evolutionary prototype that progressively integrates the implementation of each increment [18].
- *User involvement.* Project specifications are difficult to be understood during the preliminary life-cycle phases. A user-centered design increases customer satisfaction and promotes a high level of trust between the parties. Indeed, this feature focuses on constant communication and user participation at every stage of software development.
- *Component reuse.* The reuse of predefined and tested components speeds up product releases and promotes cost reduction as well as software reliability [11].
- *Formal and light documentation.* A well-defined documentation is a key feature to comply with user requirements. Moreover, formal analysis leads to clear and non-ambiguous specifications, and user involvement enables light and up-to-date documentation [1,13,21].
- *Automated schema transformation.* This feature involves the use of formal and automated transformations between schemata representing different software perspectives (e.g., between conceptual and logical schemata). This accelerates software development and promotes standard processes [13].

Table 1 summarizes the relationship between these methodological principles and the four quality goals introduced in Section 2, i.e., it gives an idea of how each principle can help increase each quality factor with specific reference to a data warehouse project. More details are given in the following section.

4 From Principles to Methodology: 4WD

In this section we propose an innovative design methodology, called *Four-Wheel-Drive* (4WD), leaning on the principles discussed in the previous section. These principles are applied in such a way as to effectively balance their pros and their cons, as resulting from practical evidences emerged during the real data warehouse projects 4WD was applied to. Besides the projects we were directly involved in, our findings are based on an elaboration of the experiences collected during the last five years by some practitioners we collaborate with.

Table 1. Expected impact of methodological principles on process quality goals

	Reliability	*Robustness*	*Productivity*	*Timeliness*
Incrementality and risk-based iteration	continuous feedback, clearer requirements	better management of change	better management of project resources, rapid feedback	early detection of errors
Prototyping	frequent tests, easier error detection		early deliveries	
User involvement	better requir. validation, better data quality			early error detection
Component reuse	error-free components		faster design	predictable development
Formal & light documentation	clearer requirements	easier evolution	faster design	
Autom. schema transformation	optimized performances	easier evolution	faster design	predictable design

As sketched in Figure 2, 4WD is based on nested iteration cycles. The external one is called *data mart cycle*; it defines and maintains the global plan for the development of the whole data warehouse and, at each iteration, it incrementally designs and releases one data mart. Data mart design is achieved by the *fact cycle*, that refines the data mart plan and incrementally designs and releases its facts[1]. Finally, fact design is based on two cycles (*modeling* and *implementation* cycles, respectively), that include the core of analysis, design, and implementation activities for delivering reports and applications concerning a single fact. The documents produced can be distinguished into releases (that correspond to project milestones) and deliveries (used for testing and validation). Remarkably, cycles are nested in a way that enables a reassessment of the decisions made during an outer iteration based on the evidences emerging from an inner iteration.

The main activities carried out in the data mart cycle are:

- *Architectural sketch*, during which the overall functional and physical architecture of the data warehouse is progressively drawn based on a macro-analysis of user requirements and an exploration of data sources as well as on budget, technological, and organizational constraints.
- *Conformity analysis*, aimed at determining which dimension of analysis will be conformed across different facts and data marts. Conforming hierarchies in terms of schema and data is a key element to allow cross-fact analysis and obtain consistent results.
- *Data mart prioritization*, based on a trade-off between user priorities and technical constraints.

[1] A fact is a concept relevant to decision-making processes, and it typically models a set of events taking place within a company (such as sales, shipments, and purchases).

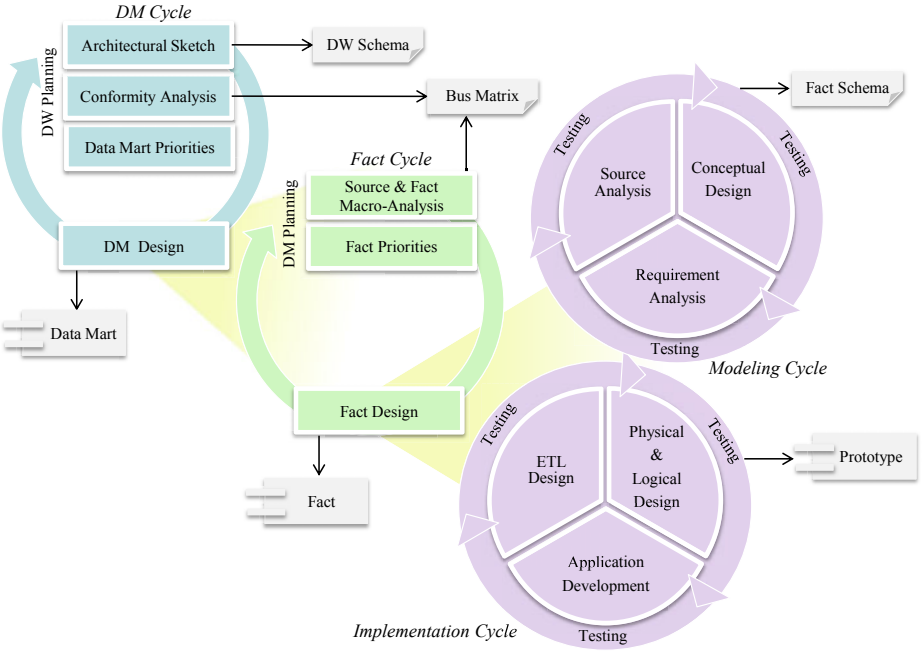

Fig. 2. A sketch of the 4WD methodology

– *Data mart design*, which builds and releases the top-priority data mart. After each data mart has been built, the three phases above are iterated to allow the data warehouse plan to be refined and updated.

The activities carried out within a fact cycle are:

– *Source and fact macro-analysis*, aimed at checking the availability, quality, and completeness of the data sources and determining the main business facts to be analyzed by users.
– *Fact prioritization* that, like for data marts, is the result of a trade-off between user requirements and technical priorities.
– *Fact design*, which develops and releases the top-priority fact. After that, the two phases above are iterated to allow the data mart plan to be refined and updated.

Finally, the activities necessary to release a single fact (or even a small set of strictly related facts) are grouped into two separate sub-cycles to emphasize that releasing a conceptual schema of a fact marks a clear separation between a modeling and an implementation phase for the fact itself. Validating the conceptual schema of a fact before implementation leads to reducing the number of implementation cycles, i.e., to faster fact cycles. While modeling should come before implementation, the activities included in each sub-cycle are not strictly sequential and can be differently prioritized by each project team. Each sub-cycle can

be iterated a number of times before its results (the conceptual schema in the first case, the analysis applications in the second) are validated and released.

In the following subsections we will discuss how 4WD meets the principles introduced in Section 3. Then, we briefly present the main outcomes of the application of 4WD to a real project in the area of pay-tvs.

4.1 Incrementality and Risk-Based Iteration

As suggested by the RAD approach, iteration is at the core of 4WD and is coupled with incremental development, that aims at slicing the system functionality into increments; in each increment, a portion of the system is designed, built, and released. Developing a system through repeated cycles leads to lower risk of misunderstood requirements (higher reliability and timeliness), to faster software deliveries (higher productivity), and to more flexible management of evolving requirements and emerging critical issues (higher robustness) [15].

Though these advantages are largely acknowledged in all modern methodologies, the type of iterations and their frequencies vary from one another depending on the type of software to be developed. For example, agile methodologies pushes segmentation to the limit by centering iteration on the so-called *user stories*, meant as high-level functional requirements —concisely expressed by users in their business language— that can be released in a few days. Since functional requirements in data warehouse projects are mainly expressed in terms of analysis capabilities, agile data warehouse design often focuses each iteration on a small set of reporting or OLAP functionalities. While this may sound natural to business users, it can lead to dramatically increasing the overall design effort, because it gives little or no relevance to the multidimensional schemata adopted to store information. Indeed, as reported by designers who adopt functionality-centered iterations in data warehouse projects, a common problem is that they fail in recognizing that apparently different analyses, designed during separate iterations, are actually supported by the very same multidimensional schema.

In 4WD, the shortest iterations that release a tangible result to users are those for modeling and implementing a single fact, that are normally completed in 2-4 weeks overall. This release rate could seem to be not very high, but it is backed by quite more frequent deliveries. Indeed, the modeling and implementation cycles have a daily to weekly frequency; the deliveries they produce enable a progressive refinement of the fact conceptual schema and implementation through a massive test based on active involvement of users.

Incremental techniques require a driver to define an order for developing increments. In 4WD this is done when deciding data mart and fact priorities, and in both cases risk is the driver —as suggested by the Spiral Software Development approach [3]. The project team should balance the risk of early releasing data marts/facts that are not highly valuable to users —which would lead users to lose interest in the project— against the risk of ordering design activities in a non-optimal way —which would determine higher costs and a longer overall project duration. Some guidelines for reducing the risk in data mart prioritization are: (a) Give priority to data marts that include widely shared hierarchies,

which makes the overall schema more robust and ensures that dimensions are fully conformed; (b) Give priority to data marts that are fed from stable and well-understood data sources; and (c) Postpone data marts based on unclear requirements, assuming that these requirement will be better understood as the user's involvement in the project increases. As to facts: (a) Give priority to facts that include the main business hierarchies and require the most complex ETL procedures; (b) Adopt a data-driven approach to design rather than a requirement-driven one whenever users do not appear to have a deep knowledge of the business domain; and (c) Plan the length of an iteration in proportion to the complexity of the fact, since failing a release in the early stage of a project will undermine the team credibility.

4.2 Prototyping

Prototyping has a crucial role in most modern software projects. In a data warehouse project, an *evolutionary* (where a robust prototype is continuously refined) and *incremental* (where the prototype is gradually enlarged by adding new subsystems) approach to prototyping is generally preferable to a *throw-away* approach (where the prototype is used to demonstrate a small set of functions and then is abandoned). In fact, the effectiveness of prototyping is maximized when the prototype is tested together with users, and in a data warehouse project this requires the whole data flow —from operational sources to the front-end through ETL— to be prototyped: a large effort, that should not be wasted. The main advantages of prototyping, with particular reference to a data warehouse project, can be summarized as follows:

- Prototypes help designers to validate requirements, because they allow users to evaluate designers' proposals by trying them out, rather than interpreting design documents. This is particularly crucial to enable a better understanding of hierarchies by users [24].
- Prototypes are especially valuable to improve the design of reports and analysis applications, due to their interactive nature. In general, prototype-based user-interfaces have higher usability [10].
- Prototypes can be used to advance testing to the early phases of design, thus reducing the impact of error corrections. For instance, an early loading test can be effectively coupled with a preliminary functional test of front-end applications to check for correct data balancing [8].
- Prototypes can be used to evaluate the feasibility of alternative solutions during logical design of multidimensional schemata and during ETL design. This typically leads to improved performance and maintainability, and to reduced development costs [24].

The above points are basically associated with an increase in reliability and productivity. More specifically, the impact on reliability is related to both data schemata, data quality, and performances. First of all, having a working prototype available during the early project phases enables the designer to keep a strict and constant control over the data schema to ensure that it fully supports

user requirements. Then, data quality can be improved by closely involving users in testing the prototype using both real and ad-hoc generated data. Finally, an incremental approach can also be used to take better care of performance issues by following the modularity principle to separate correctness from efficiency. This means that a working prototype can be delivered first; then, performances can be improved during the following iteration to deliver an increment in the form of a working *and efficient* prototype.

4.3 User Involvement

Recent years have been characterized by a growing awareness that human resources are one of the keys to a project success. In this direction, some modern software design methodologies tend to emphasize organizational factors rather than technical aspects. For instance, agile approaches pursue the idea of creating responsible and self-organizing teams to maximize participation of developers and their productivity. They also focus on user involvement as a means to reduce the risk of expressing ambiguous requirements and make software validation easier and more effective [1].

4WD pays a large attention to user involvement because it has a substantial influence on process reliability and timeliness. User involvement can be promoted in different ways:

- All users should preliminary receive a comprehensive training to clarify the project goals, explain the multidimensional model, and introduce a shared language for conceptual design.
- Prototyping is the most effective way to have users participate in the design process and keep them aware of the project status.
- Due to the complex data transformation that is inherent to data warehouse systems, only users —who have insight of business data— can easily detect problems and errors. So, most testing activities should be based on user feedback. User involvement is specifically crucial for usability tests of reporting and OLAP front-ends, and for functional tests of ETL procedures.

4.4 Component Reuse

Applying a component-based methodology means using predefined elements to support the software development process [11]. This is often done by data warehouse designers, though mostly in an unstructured way. The components that can most effectively be reused in a data warehouse project are:

- Conformed hierarchies, that are reused in different facts and data marts. Using conformed hierarchies not only accelerates conceptual design, but is also the key for achieving an enterprise view of business in a data warehouse.
- Library hierarchies, that model common hierarchy structures for a given business domain. For instance, a customer hierarchy in a sales analysis has some basic features that can be easily reused in different data warehouse projects to reduce the effort in designing facts.

- Library facts, that define common measure and dimension structures as emerging from design best practices for a given business domain. Of course, library facts must be tailored to specific user needs; nevertheless, they may be very useful in requirement-driven approaches to give designers and users a starting point for conceptual design.
- ETL building blocks, meant as predefined extraction, transformation, cleaning, and loading routines (e.g., a routine for cleaning a geographical attribute against the list of ISO 3166-2 codes for administrative divisions, or one for loading a type-3 slowly-changing dimension from an operational data store). Reusing such routines reduces the ETL design effort and makes ETL more reliable due to the use of largely-tested algorithms.
- Analysis templates, that define a reference structure for reports and applications. In particular, sharing an analysis template across a data warehouse project is warmly suggested to standardize the interface presented to users.

4WD takes advantage of component reuse to accelerate development and increase robustness. While ETL tools already include some building blocks that can be easily reused through parameterization, identifying hierarchies and facts to be reused deserves more attention. 4WD devotes an ad-hoc phase (*conformity analysis*) to identifying hierarchies to be conformed using a bus matrix. Besides, conceptual schemata are a very effective tool to formalize the structure of facts and hierarchies and support their matching against the available libraries.

4.5 Formal and Light Documentation

In waterfall approaches, documentation is extensively used during the whole life-cycle to support the design process and represent and validate requirements. Other approaches, like RAD and agile methodologies, tend to discourage the use of documentation (other than the one automatically produced by tools) because it may lead to prematurely freezing requirements and slowing down iterations, and suggest to replace it with continuous communication with users [1,15].

While we agree that textual documentation should be reduced to the minimum, we firmly believe that formal documentation is a key factor to promote precise formalization of requirements, clear communication between designers and users, accurate design, and maintainability. In 4WD, the main role to this end is played by conceptual schemata. In particular:

- At the data warehouse level, we mostly use a simple but effective schema that summarizes the data marts, their data sources, and the profiles of the users who access them [9]. This high-level schema is first drawn during the architectural sketch phase, and refined after each data mart cycle. It is essentially used to share the basic functional architecture with users and to support the discussion of data mart priorities.
- At the data mart level, an important role is played by a *bus matrix* that associates each fact with its dimensions, thus pointing out the existence of

conformed hierarchies. This schema is built and progressively refined during the *conformity analysis* and *fact macro-analysys* phases, and is used to test that the designers has properly captured the existing similarities between different facts and different data marts, thus ensuring their integrability [9].
- At the fact level, we force designers to complete and release the conceptual schema of a fact *before* proceeding with implementation. Indeed, having users and designers clearly agree on the fact granularity and measures, as well as on the hierarchy structures and semantics, is the most effective way to avoid misunderstandings and omissions. Finding this agreement informally, or leaning on the logical/physical schema of the fact, is obviously hard and error-prone, while a (graphical) conceptual schema is clearly understood even by non-technical users. In particular, we adopted the Dimensional Fact Model [9] in a number of projects for public administrations (such as local health authorities, the Ministry of Justice, the State Accounting Department) and we verified that fact schemata are also understood by non-IT people such as physicians and jurists.

A major role in this context is also played by metadata, that multidimensional engines store to describe the structure of a data mart. Metadata can typically be exported to generate a documentation based on standard languages (such as XML) and models (such as the CWM); this also encourages interoperability, that is normally seen as a crucial issue in data warehouse projects.

4.6 Automated Schema Transformation

To reduce design complexity, the MDA approach proposes to use formal models for separately specifying a *Platform Independent Model* (PIM, it represents system functionalities at a conceptual level) and a *Platform Specific Model* (PSM, it gives a logical and platform-dependent representation of system functionalities), and to use automated transformations to derive a PSM from a PIM. In a data warehouse project, this can be applied to design both ETL procedures and multidimensional schemata, as shown in [23,16].

In 4WD, automated schema transformations are encouraged, mainly to speed up design and simplify evolution, as long as they need a reasonable effort from users to understand formal models and they do not require to invest too many resources in activities that are not directly valuable to users. We propose two metadata-based activities for automation, possibly supported by CASE tools:

- Supply-driven conceptual design. In supply-driven approaches, a basic conceptual schema for a fact can be automatically derived starting from the logical schema of operational data sources [17]. When applicable, this is a very effective way to cut design costs.
- Logical design. A logical schema can be automatically obtained from a conceptual schema by applying a set of transformations that express common design rules and best practices, possibly based on the expected workload [9].

4.7 Practical Evidences

4WD was applied to a project in the area of pay-tvs. The project had an overall duration of 6 months and was carried out by an Italian system integrator specialized in BI applications.

During data warehouse planning two data marts were identified, namely administration and management control, that were prioritized according to their importance for users: the administration data mart was given higher priority because its size is definitely larger (9 vs. 3 facts). During data mart planning we organized the overall project in 7 releases (5 for the first data mart, 2 for the second one), each centered on at most 3 facts and taking from 10 to 26 days. Facts were grouped into a single release when they either shared several dimensions or had similar ETL processes (e.g., because measures were extracted from the same data sources and tables), as emerging from conformity analysis and source and facts macro-analysis. Each release was then assigned a value from the users point of view, an estimated nominal complexity, and a risk expressed as a percentage complexity overhead (ranging from 19 to 35%) to determine a worst-case complexity. The criteria used for establishing release priorities were: (1) advance the most valuable facts to early releases; (2) uniformly distribute the worst-case complexity; and (3) respect the dependencies in fact implementation. Besides, some fact were delayed because the development of specific extraction interfaces by external consultants was required for some of their source data; other facts were postponed due to some uncertainty on the requirements. After each release, its actual duration was compared to the estimated complexity. In 2 cases it turned out that the estimation was inaccurate; this was fixed right away by revising the remaining estimates and by changing the team composition.

One of the benefits of adopting 4WD in this project was the speed-up due to large user involvement and extensive prototyping. Users were enabled to access a web portal to signal the errors, and monitor the team's answers and the project state. This was particularly effective for improving the structure of reports and the business rules for detecting source data errors. Noticeably, all errors signaled by users were related to wrong data: user mainly own empirical knowledge, so it may be hard for them to reason from an abstract point of view (e.g., to evaluate an ETL flow or a report structure with no data loaded). The implementation effort was reduced by partially reusing existing reports and dimension tables, because those required by administration and management control users are quite standard. This was not the case for ETL, that required a strong personalization, so reuse was limited to some basic routines made available by the adopted ETL suite. Finally, adopting the DFM as a conceptual model enabled designers to produce a concise but exhaustive documentation, and to use a CASE tool to automate logical design [7].

5 Related Literature and Discussion

In this paper we started by identifying the main problems behind data warehouse projects, and we ended up with proposing an original methodology, 4WD,

inspired by six basic principles of modern software engineering. In this section we critically compare 4WD with the existing data warehouse design methodologies.

Data warehouse design has been investigated by the research community since the late nineties. A classic waterfall approach was first proposed in [6]; a distinguishing feature was the inclusion of a conceptual design phase aimed at better formalizing the data schema. A sequential approach to design is also followed in [14], where an object-oriented method based on UML is proposed to cover analysis, design, implementation, and testing. Another UML-based method is presented in [19]; here, the use of the *Common Warehouse Metamodel* (CWM) is suggested to promote a more standard approach to conceptual design. All these methodologies follow a linear approach that hardly adapts to changes and is unsuitable when requirements are uncertain. In 4WD these problems are overcome thanks to iteration and prototyping.

Iterative solutions are typically adopted by methodologies like RAD and Agile. The work in [12] breaks with strictly sequential approaches by applying two Agile development techniques, namely *scrum* and *eXtreme Programming*, to the specific challenges of data warehouse projects. To better meet user needs, the work suggests to adopt a *user stories decomposition* step based on a set of architectural categories for the back-end and front-end portions of a data warehouse. However, it does not deeply discuss how this decomposition impacts on modeling and design. In this direction, 4WD emphasizes the key role of the multidimensional model as a driver for the development process and promotes fact-based iterations to increase its productivity while preserving reliability.

A different approach to tackle the data warehouse design complexity is the MDA methodology proposed in [16] to better separate the system functionality from its implementation. Strong relevance is given to the development of the data warehouse repository; the three main perspectives of MDA (CIM, PIM, and PSM) are defined using extensions of UML and CWM, and the inter-model transformations are described using the *Query/View/Transformation* (QVT) language. In practice, strictly applying this methodology may be hard due to the poor aptitude of users for reading formal models and investing resources in low-values activities. To overcome these issues, in 4WD automation is specifically targeted on supply-driven conceptual design and logical design. This reasserts the key role played by conceptual schemata of facts in 4WD.

A pragmatic comparison between data warehouse design methodologies is offered in [22], where 15 different solutions proposed by Business Intelligence software vendors are examined. The authors emphasize the lack of software-independent approaches, and point out that all the proposed solutions hardly can deal with changes and market evolution, which creates a robustness problem. To improve robustness, 4WD specifically relies on three key factors: (a) iteration breaks the linear development process by offering frequent deliveries and reviewing points; (b) a formal and light documentation provides a clear picture of the current specifications, facilitating the identification of the units to be evolved; (c) automating schema transformations reduces the time needed to propagate changes to the different levels.

References

1. Agile Manifesto: Manifesto for agile software development (2010), http://agilemanifesto.org/
2. Beyer, M., Richardson, J.: Agile techniques augment but do not replace business intelligence and data warehouse best practice. Tech. Rep. G00201031, Gartner Research (2010)
3. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer 21(5), 61–72 (1988)
4. Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of software engineering. Prentice Hall, Englewood Cliffs (2002)
5. Giorgini, P., Rizzi, S., Garzetti, M.: GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems 45(1), 4–21 (2008)
6. Golfarelli, M., Rizzi, S.: A methodological framework for data warehouse design. In: Proc. DOLAP, pp. 3–9 (1998)
7. Golfarelli, M., Rizzi, S.: WAND: A CASE tool for data warehouse design. In: Proc. ICDE, pp. 7–9 (2001)
8. Golfarelli, M., Rizzi, S.: A comprehensive approach to data warehouse testing. In: Proc. DOLAP, pp. 17–24 (2009)
9. Golfarelli, M., Rizzi, S.: Data warehouse design: Modern principles and methodologies. McGraw-Hill, New York (2009)
10. Gordon, V.S., Bieman, J.M.: Rapid prototyping: Lessons learned. IEEE Software 12(1), 85–95 (1995)
11. Heineman, G.T., Councill, W.T.: Component-based software engineering: Putting the pieces together. Addison-Wesley, Reading (2001)
12. Hughes, R.: Agile Data Warehousing: Delivering world-class business intelligence systems using Scrum and XP. IUniverse (2008)
13. Kruchten, P.: The 4+1 view model of architecture. IEEE Software 12(6), 42–50 (1995)
14. Luján-Mora, S., Trujillo, J.: A comprehensive method for data warehouse design. In: Proc. DMDW (2003)
15. Martin, J.: Rapid application development. MacMillan, Basingstoke (1991)
16. Mazón, J.N., Trujillo, J.: An MDA approach for the development of data warehouses. In: Proc. JISBD, pp. 208–208 (2009)
17. Moody, D., Kortink, M.: From enterprise models to dimensional models: A methodology for data warehouse and data mart design. In: Proc. DMDW (2000)
18. Pomberger, G., Bischofberger, W.R., Kolb, D., Pree, W., Schlemm, H.: Prototyping-oriented software development — concepts and tools. Structured Programming 12(1), 43–60 (1991)
19. Prat, N., Akoka, J., Comyn-Wattiau, I.: A UML-based data warehouse design method. Decision Support Systems 42(3), 1449–1473 (2006)
20. Ramamurthy, K., Sen, A., Sinha, A.P.: An empirical investigation of the key determinants of data warehouse adoption. Decision Support Systems 44(4), 817–841 (2008)
21. Royce, W.W.: Managing the development of large software systems: Concepts and techniques. In: Proc. ICSE, Monterey, California, USA, pp. 328–339 (1987)
22. Sen, A., Sinha, A.P.: A comparison of data warehousing methodologies. Commun. ACM 48(3), 79–84 (2005)
23. Simitsis, A., Vassiliadis, P.: A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decision Support Systems 45(1), 22–40 (2008)
24. Sommerville, I.: Software Engineering. Pearson Education, London (2004)