Chapter IX
Applying Agility to Database Design

Guoqing Wei
FedEx Corporation, USA

Linda Sherrell
University of Memphis, USA

APPLYING AGILITY TO DATABASE DESIGN

Agile methods are flexible, allowing software developers to embrace changes during the software development life cycle. But the introduction of agile practices into software organizations may cause unhealthy tensions between the developers and data professionals. The underlying reason is that when agile methodologies are employed, the two communities use incompatible approaches, namely simple design and iterative development, which are practices associated with all agile methodologies, and big design up front (BDUF), a popular database technique. BDUF is inflexible, as once the database foundation is set, it is difficult to make changes throughout the software development life cycle. This chapter describes a database development method for a Web environment. Using this method, a data professional divides the database into loosely coupled partitions and resolves the above conflicts by applying certain agile practices. The result is that the database development becomes more iterative and incremental. This has the added benefit of supporting rapid application development in a dynamic environment, a fundamental characteristic of most Web applications.

BACKGROUND

Traditional Software Engineering

Traditional software life cycle models, also known as plan-driven or document-driven approaches, treat software development as a linear planning process where the majority of the planning is completed before design. The waterfall model, which consists of five phases, as shown in Figure 1, is the most common example. In the requirements phase, developers aim to determine their
Applying Agility to Database Design

Figure 1. waterfall model

requirements change

requirements change

requirements change

Figure 1. waterfall model

requirements change

Figure 1. waterfall model

requirements change

requirements change

Figure 1. waterfall model

requirements change

requirements change

clients’ needs based on interviews or questionnaires. Then in the analysis phase, developers analyze and refine the requirements and amend their findings in a software requirements specification (SRS) document, which acts as a contract between the developers and clients. Once the SRS is agreed upon, developers move to the design phase. The implementation phase does not start until completion of the overall design. After the product implementation phase, product delivery to the clients occurs, and finally the software development lifecycle moves to the last phase, post-delivery maintenance (Schach, 2003).

About 40% of software development companies still use the traditional waterfall model (Neil & Laplane, 2003). The document-driven approach is seen as an advantage for large, complex projects such as safety critical software or aerospace projects where there may be multiple contractors. The associated standards with their predictability help to mitigate personnel turnover and allow management to better compare projects (Boehm & Turner, 2004).

However, there are several drawbacks to the waterfall model. In particular, when the requirements change, the developers have to stop the current analysis, design or implementation, return to the requirements phase, and renegotiate the SRS. This process may also involve a change control board. The development can not continue until the requirements have been finalized again; this consumes time and resources. Another drawback is that clients do not see the product until the development organization finishes the implementation and releases the product. Furthermore the different background knowledge between developers and clients can also lead to alternative understandings of the SRS with the final product often not meeting the client’s expectations.

To help counteract the limitations of the waterfall model, the rapid prototyping software life cycle model may be employed. In particular, this model allows feedback from the customer during requirements analysis, so there is a better likelihood that the final product will meet the customer’s needs. Software developers may even decide to use an incremental model in order to gain feedback from the customer multiple times in the software life cycle. Note that this model still requires extensive planning of the overall architectural design before increments are implemented (Schach, 2003).