CHAPTER 1

Leading-Edge Software Development

Modern software development requires modern ways of working.

The only constant in the information technology (IT) industry is change. To remain employable, let alone effective, software developers must continually take the time to identify and then understand the latest development approaches. The goal of this chapter is to introduce you to leading-edge technologies and techniques that enable you to succeed at developing modern business systems. I will try to steer you through the marketing hype surrounding these approaches, and in one case try to dissuade you from adopting it—just because something is new and well hyped does not mean that it has much of a future. In short, this chapter provides you with a foundation for reading the rest of this book.

This chapter discusses

- Modern development technologies;
- Modern development techniques;
- How this book is organized; and
- The case studies.

1.1 Modern Development Technologies

Effective developers understand the fundamentals of the technologies that they have available to them. The good news is that we have many technologies available to us; the bad news is that we have many technologies available to us.

Figure 1.1, which depicts a high-level architecture detailing how these technologies are used together, shows how some applications may be *n*-tiered—an approach where application logic is implemented on several (*n*) categories of computing devices (tiers)—whereas others fall into the "fat client" approach where most business logic is implemented on the client. Object technology is used to implement all types of logic, including both business and system logic. XML is used to share data between tiers, and Web services are used to access logic that resides on different tiers. Most business data are stored in relational databases, which are accessed either via structured query language (SQL) or persistence frameworks (see Chapter 14). Data are returned from the database as a collection of zero or more records and then marshaled either into objects or into XML documents. In the case of a browser-based application the XML structures are in turn converted into HTML documents, often through XSL-T (extensible stylesheet language transformations).

Although the focus of this book is the development of business systems, much of the advice is also applicable to the development of other types of software. My specialty is business software so that is what I will stick to in this book. My experience is that when it comes to building modern business systems, you are very likely to use a combination of the following:

- Object technology;
- Extensible markup language (XML);
- Relational database (RDB); and
- Web services.

1.1.1 Object Technology

The object-oriented (OO) paradigm (pronounced "para-dime") is a development strategy based on the concept that systems should be built from a collection of reusable parts called objects. Examples of OO languages and

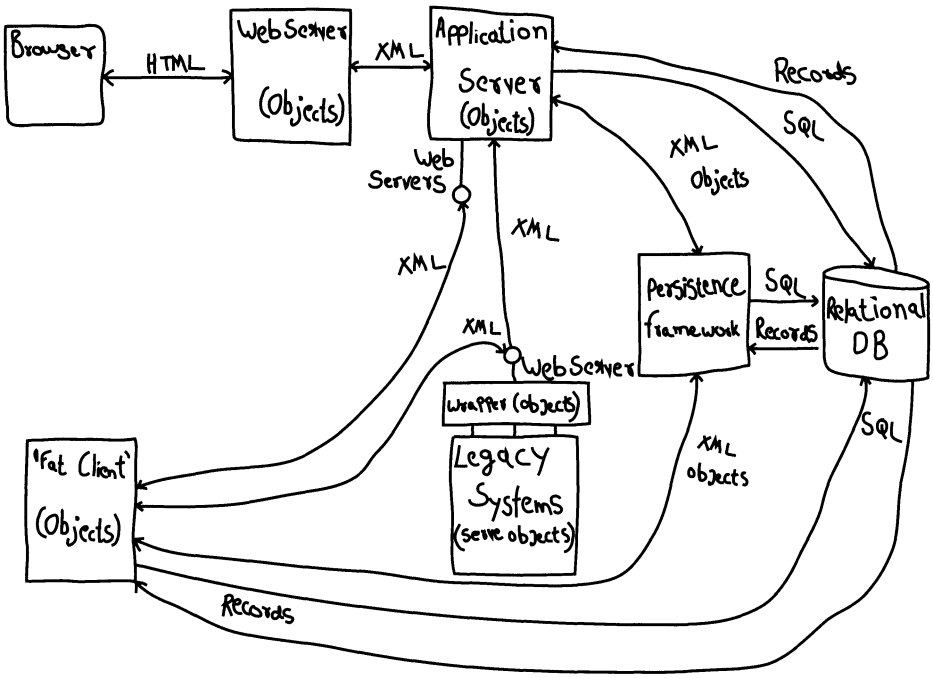

FIGURE 1.1. High-level application architecture.

technologies include the Java, C#, and C++ programming languages and the Enterprise JavaBeans (EJB) framework. The original motivation of the object paradigm was that objects were meant to be abstractions of real-world concepts, such as students in a university, seminars that students attend, and transcripts that they receive. This was absolutely true of business objects, but as you will see throughout this book, business objects are only one part of the picture—you also need user interface objects to enable your users to work with your system, process objects that implement logic that works with several business concepts, system objects that provide technical features such as security and messaging, and potentially some form of data objects that persists your business objects.

The use of object technology can be in fact quite robust. In fat client applications object technology is typically used on client machines, such as personal computers or personal digital assistants (PDAs), to implement both user interface code and complex business logic. In thin-client or *n*-tier applications object technology is often used to implement business logic on application

TABLE 1.1. Evaluating Object Technology

Advantages	Disadvantages
• Enables development of complex software • Wide industry acceptance • Mature, proven technology • Wide range of development languages and tools to choose from • Very easy to find people with object experience	• Significant skillset is required • No single language dominates the landscape (although Java, C#, C++, and arguably Visual Basic are clearly popular and here to stay) • Not all IT professionals, in particular some within the data community, accept it • Technical "impedance mismatch" with structured technologies and RDBs

servers and sometimes even on other nodes such as database servers, security servers, or business rule servers. Table 1.1 summarizes the strengths and weaknesses of object technology for business system development.

It is useful to contrast these concepts with the structured paradigm and structured technology. The structured paradigm is a development strategy based on the concept that a system should be separated into two parts: data (modeled using a data model) and functionality (modeled using a process model). Following the structured approach, you develop applications in which data are separate from behavior both in the design model and in the system implementation (that is, the program). Examples of structured technologies include the COBOL and FORTRAN programming languages. The main concept behind the object-oriented paradigm is that instead of defining systems as two separate parts (data and functionality), you now define systems as a collection of interacting objects. Objects do things (that is, they have functionality) and they know things (they have data). While this sounds similar to the structured paradigm, in practice it actually is quite different.

However, it is equally important to recognize that structured techniques and technologies still have their place. As you will see later in Section 1.1.4, it is quite common to transform legacy systems, typically implemented with structured technologies, and then wrap them with Web services to reuse their functionality. Furthermore, in coming chapters you will discover that structured

```
<office>
  <name>Ronin International, Inc. HQ</office:name>
  <state>
    <name>Colorado</state:name>
    <area>North West</state:area>
  </state>
  <country>United States of America</country>
</office>
```

FIGURE 1.2. An example of an XML document.

modeling techniques such as data flow diagrams (DFDs) and data models are still critical to your success.

1.1.2 Extensible Markup Language (XML)

XML is a subset of standard generalized markup language (SGML), the same parent of hypertext markup language (HTML). The critical standards are described in detail at the World Wide Web Consortium Web site (http://www.w3c.org). XML is simply a standardized approach to representing text-based data in a hierarchical manner and for defining metadata about the data. From a programmer's point of view XML is a data representation, backed by metadata, plus a collection of standardized technologies for parsing that data. The data are stored in structures called XML documents and the metadata are contained in document-type definitions (DTDs) and XML schema definitions. Figure 1.2 provides an example of a simple XML document and Table 1.2 overviews the advantages and disadvantages of XML.

XML is often used to transfer data within an application when that application has been deployed across several physical servers. It is also used in enterprise application integration (EAI) as a primary means of sharing data between applications. You will also see XML used for permanent storage in data files; it is quite common to use XML for configuration files in both J2EE and .NET applications, and sometimes even in databases. Chapter 14 discusses database issues in more detail, and you will see at that point that it is often better to "shred" an XML document into individual columns instead of saving it as a single column when storing data in a relational database.

TABLE 1.2. Evaluating XML Technology

Advantages	Disadvantages
• XML is widely accepted • XML is cross platform • (Small) XML documents are potentially human readable • XML is standards based; the World Wide Web Consortium defines and promotes technical standards for XML and XML.org (http://www.xml.org) promotes vertical XML standards within specific industries • XML separates content from presentation	• XML documents are very bulky, causing performance problems • XML requires marshaling (conversion of XML to objects and vice versa), causing performance problems • XML standards are still evolving • XML is overhyped, resulting in unrealistic expectations • XML business standards will prove elusive because most businesses compete and do not collaborate with their industry peers

1.1.3 Relational Database (RDB) Technology

A relational database is a persistent storage mechanism that stores data as rows in tables. Most relational databases enable you to implement functionality in them as stored procedures, triggers, and even full-fledged Java objects. Although other alternatives to RDBs exist—object-oriented database management systems (OODBMSs), XML databases (XDBs), and object-relational databases (ORDBs)—the fact is that RDBs are the database technology of choice for the vast majority of organizations. Table 1.3 summarizes the pros and cons of using RDBs for modern business applications.

It is important to understand that there is a technical impedance mismatch between RDBs and other common implementation technologies. When it comes to RDBs and objects, RDBs are based on mathematical principles, whereas objects are based on software engineering principles (Ambler 2003a). The end result is that you need to learn how to map your objects into RDBs as well as how to use the two technologies together, the topic of Chapter 14. Similarly, there is a difference between XML and RDBs—XML structures are hierarchical trees, whereas RDB table structures are "bushier" in nature.

Table 1.3. Evaluating RDB Technology

Advantages	Disadvantages
• Wide industry acceptance • Very easy to find RDB expertise • Mature industry dominated by several strong vendors (Oracle, IBM, Sybase, Microsoft) • Open source databases, for example, MySQL, are available • Wide range of development tools • Sophisticated and flexible data processing are supported	• Impedance mismatch with other common technologies, in particular objects and XML

This requires you either to write marshaling code that maps individual XML elements to table columns, degrading performance, or to simply store XML documents in a single column, negating many of the benefits of RDBs.

1.1.4 Web Services

According to the World Wide Web Consortium, a Web service is "a software application identified by a Uniform Resource Identifier (URI), whose interface and bindings are capable of being identified, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based protocols." Whew! An easier definition is that a Web service is a function that is accessible using standard Web technologies in accordance to standards (McGovern et al. 2003). The Web Services Interoperability Organization (WS-I, http://www.ws-i.org) is a consortium of mostly vendor companies that focus on Web services standards.

Web services are being used to implement functionality that is accessible via Internet technologies, often following an approach referred to as utility computing where the use of a computing service is charged for by the vendor on a usage basis much as electricity or water is charged for. It is far more common to use Web services "behind the firewall" to implement reusable functionality or to wrap legacy systems, including both programs and databases, so that they may be reused by other applications. Internal Web services such as this

TABLE 1.4. Evaluating Web Service Technology

Advantages	Disadvantages
• Promotes reusability through a standardized approach • Supports location transparency through a UDDI server • Contains scaleable architecture • Enables you to reduce dependency on vendors	• Searching for services via UDDI is time consuming • Overhead of XML detracts from performance • Does not yet support transaction control (this is coming) • Does not yet support security (this is coming)

are often managed within an internal UDDI (universal description, discovery, and integration) registry or better yet a reuse repository such as Flashline (http://www.flashline.com). A system built from a collection of cohesive services has a service-oriented architecture (SOA). Table 1.4 summarizes the advantages and disadvantages of Web services.

1.2 MODERN DEVELOPMENT TECHNIQUES

Now that we understand the fundamentals of modern technologies, we should now consider modern development techniques. The IT industry is currently undergoing what I consider to be a significant shift—a move from prescriptive development techniques to agile techniques. Until just recently management often bemoaned the fact that developers did not want to follow a process, not understanding what was wrong with the 3,000 pages of procedures they expected everyone to follow. Along came agile software processes such as extreme programming (XP) (Beck 2000), feature-driven development (FDD) (Palmer and Felsing 2002), and agile modeling (Ambler 2002) and developers embraced them. Unfortunately many managers are still leery of agile techniques and fight adoption of them. This is a truly ironic situation—developers are now demanding to follow proven software processes yet are not being allowed to do so. Sigh.

In this section I briefly explore four important development techniques that all developers should be familiar with:

- Agile software development;
- Unified modeling language (UML);
- The unified process; and
- Model-driven architecture (MDA).

1.2.1 Agile Software Development

Over the years several challenges have been discovered with prescriptive software development processes, such as the waterfall lifecycle characterized by the ISO 12207 standard (http://www.ieee.org), the Object-Oriented Software Process (OOSP) (Ambler 1998b, 1999), and the Rational Unified Process (RUP) (Kruchten 2000). First, the Chaos report published by the Standish Group (http://www.standishgroup.com) still shows a significant failure rate within the industry, indicating that prescriptive processes simply are not fulfilling their promise. Second, most developers do not want to adopt prescriptive processes and will find ways to undermine any efforts to adopt them, either consciously or subconsciously. Third, the "big design up front" (BDUF) approaches to software development, particularly those followed by ISO 12207, are incredibly risky due to the fact that they do not easily support change or feedback. This risk is often ignored, if it is recognized at all, by the people promoting these approaches. Fourth, most prescriptive processes promote activities only slightly related to the actual development of software. In short, the bureaucrats have taken over.

To address these challenges a group of 17 methodologists formed the Agile Software Development Alliance (http://www.agilealliance.org), often referred to simply as the Agile Alliance, in February 2001. An interesting thing about this group is that they all came from different backgrounds, and yet they were able to come to an agreement on issues that methodologists typically do not agree upon. They concluded that to succeed at software development you need to focus on people-oriented issues and follow development techniques that readily support change. In fact, they wrote a manifesto (Agile Alliance 2001a) defining four values for encouraging better ways of developing software:

1. **Individuals and interactions *over* processes and tools.** The most important factors that you need to consider are the people and how they work together because if you do not get that right the best tools and processes will not be of any use.

2. **Working software *over* comprehensive documentation.** The primary goal of software development is to create software, not documents—otherwise it would be called documentation development. Documentation has its place; written properly it is a valuable guide for people's understanding of how and why a system is built and how to work with the system.
3. **Customer collaboration *over* contract negotiation.** Only your customer can tell you what they want. They likely do not have the skills to exactly specify the system, they likely will not get it right at first, and they will likely change their minds. Working together with your customers is hard, but that is the reality of the job. Having a contract with your customers is important, but a contract is not a substitute for communication.
4. **Responding to change *over* following a plan.** Change is a reality of software development, a reality that your software process must reflect. People change their priorities for a variety of reasons, their understanding of the problem domain changes as they see your work, and the business environment changes, as does technology. Although you need a project plan, it must be malleable and it can be in fact very simple (unlike many of the Gantt charts you may have seen in the past).

The important thing to understand is that while you should value the concepts on the right-hand side you should value the things on the left-hand side even more. A good way to think about the manifesto is that it defines preferences, not alternatives, encouraging a focus on certain areas but not eliminating others.

The Agile Alliance also defines a collection of twelve principles (Agile Alliance 2001b), based on the four values:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.