Books

SanFrancisco Component Framework: An Introduction, Paul Monday, James Carey, and Mary Dangler, Addison-Wesley Longman, Inc., Reading, MA, 2000. 338 pp. (ISBN 0-201-61587-8).

IBM's SanFrancisco* component framework is the topic of this book, written to provide software developers and managers with a short and readable introduction to using this enterprise framework to conceive and build business-oriented applications. The product is the result of the joint work of IBM and independent software vendors to define a customizable, object-oriented framework as a standard platform to create, configure, and manage different kinds of applications ranging from ledger control to a warehousing system.

For many years existing legacy systems have had several common problems, as many developers know: large parts of applications often needed to be re-engineered, maintenance issues and dynamic business requirements presented many challenges, and people needed constant retraining. The contribution of SanFrancisco, in my opinion, is that it offers an approach that follows the line of the current object-oriented methodologies (Unified Modeling Language [UML] Objectory and Catalysis) and improves this process by providing the benefits associated with framework technologies.

SanFrancisco Component Framework contains fifteen well-written chapters divided into five sections, providing different levels of detail about SanFrancisco as the reader progresses. The first chapters describe the overall architecture of SanFrancisco and how it makes use of object-oriented development. This part is good for readers interested primarily in a brief and complete description of SanFrancisco. Of course, a better understanding requires reading the rest of the book. Subsequent chapters give a deeper insight into the layers that form the framework core, examples of usage, common processes (patterns), and programming of client applications. This content is of value to many audiences (analysts, designers, implementors, project managers, etc.), explaining the rationale of the framework and outlining its advantages in the new software development world that, in my opinion, is evolving.

A remarkable strength of the book is its clear educational purpose, something difficult to find in much of the literature that treats the documentation and usage of frameworks. Not only are technical aspects of SanFrancisco covered, but simple examples, conceptual diagrams, and lists of desired tasks that take real advantage of the framework facilities are explained in a comprehensive and reasonable way.

The key point for developers is to understand the layered style that organizes the SanFrancisco framework and the IBM aims that are reflected in this architecture: to provide isolation of specific technologies, allow integration with legacy systems, and provide the core of the solution that would have to be supplied by any application in a particular domain (especially a business domain). The SanFrancisco architectural vision comprises: the core business process layer (the most specific layer), which captures the main classes and processes for a particular application domain (i.e., general ledger, accounts receivable and accounts payable, order management, and warehouse management); the middle layer, the common business objects layer, consisting of classes, processes, and mechanisms that are common across many application domains (i.e., companies, currency, customers, banks, accounts, financial calendar, and generalized mechanisms); and the bottom layer, the foundation and utilities layer, which provides the basic services needed by a business application and isolates applications from the underlying technologies. Finally a Java** virtual machine is the "basement" that ensures portability and isolation of hardware platforms. Following the design features of a layered style, each of these layers offers services with a well-defined interface to its upper layer and uses services provided by its lower layer. This property has a close

relationship with reusability. Furthermore, IBM achieves another important goal: layers can be easily extended and replaced in the future, while preserving the same interface with other layers.

An important note should be made at this point. A given application need not necessarily be built on the core business process layer; developers are free to decide on what layer they choose to build their particular applications (and this is not a minor issue). A wide range of applications can be built using this powerful and robust kernel, but unfortunately it does not seem to be easy to achieve for novice users because of the need for specific knowledge and training. This aspect is clearly beyond this introductory book, and the interested reader should look for related materials.

Other useful features of the book include documentation techniques—applied to every phase of development such as requirements gathering, scenario development, and analysis and design—that help to train developers to identify requirements fulfilled in the SanFrancisco framework; availability of scenarios, class diagrams, and implementations of the scenarios; and textual template descriptions combined with UML diagrams, Javadoc tools, and getting-started tutorials. The framework supports a set of generalized mechanisms (based on the design patterns described by Gamma, Helm, Johnson, and Vlissides, the Gang of Four) that capture and specify common procedures and allow the application of these well-engineered solutions to several problems, thus maximizing reuse. The coverage and discussion of these facilities is generally good, although the SanFrancisco vision about patterns is not very easy to understand at a first reading. Perhaps because the authors refer to patterns from different points of view at the same time, this is often a source of confusion.

Although the contents of the book are mainly centered in server-side aspects, the last chapters of *SanFrancisco Component Framework* address some issues related to client applications, taking briefly into account the type of client and interface needed, stand-alone applications, Web-based systems, multithreading, and JavaBeans** components, among other subjects. Some knowledge of the SanFrancisco programming model is required for developers to build a well-behaved application, but developers are not restricted to SanFrancisco alone, and other commercial products can be used and integrated.

With respect to building graphical user interfaces (GUIs) in the presentation layer, SanFrancisco offers a full Model-View-Controller architecture that achieves a component separation that does not exist in the Java Abstract Window Toolkit (AWT), giving to client application developers all the benefits that this classical style provides. It is my impression that the controllers (called "maintainers" in SanFrancisco terminology) seem to be overloaded in functionality in the SanFrancisco approach, and it is difficult to understand the relationship that appears to connect view and model in the diagram of the Model-View-Maintainer depicted in Chapter 13. Some of the conceptual issues surrounding these misleading aspects would benefit from revision.

Overall, I recommend that any developer or manager involved in business applications read (or at least skim) this book, which offers a valuable set of experiences, design guidelines, and framework support, as well as an updated view of, and new trends in, software engineering practice. The SanFrancisco framework aims to bridge the gap between theory and practice and address reuse of both large and common patterns of behavior, reducing the "time to market" for business applications. These goals appear to be very ambitious in the software engineering domain, but I believe they are also the foundational reasons for its existence.

Mohamed E. Fayad
Computer Science & Engineering
University of Nebraska
Lincoln
Nebraska

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc.

SanFrancisco Life Cycle Programming Techniques, Maynard Johnson, Randy Baxter, and Tore Dahl, Addison-Wesley Longman, Inc., Reading, MA, 2000. 210 pp. (ISBN 0-201-61658-0).

In the business world, many processes have several stages through their life cycle (a beginning, an intermediate state or states, and an end), and certain rules are used to decide when a process is ready to move from one stage to another. The relationships between data and control are far from trivial in software organizations. In software applications, data

and control are traditionally closely coupled, and this fact leads to difficult maintenance and customization tasks. Use of the so-called SanFrancisco* Life Cycle pattern to achieve a clear separation of these parts—life-cycle control, state information, and business data—is the focus of the techniques presented in this book. The state-like pattern is defined within the SanFrancisco framework as a common process that developers can recognize and apply in different contexts, having the benefits of well-proven designs and the ability to shorten the development cycle.

"Life Cycle pattern" refers to data-driven processes used to achieve state changes. An application that is used to perform a business process that progresses from one well-defined state to another is a natural candidate to be defined in terms of a Life Cycle pattern. Basically, a Life Cycle pattern comprises the following business concepts: (1) a state machine that is able to recognize patterns of conditions and take a predefined action for each situation it recognizes, such as modify behavior of components, invoke a method, or move the processing focus from one business task to another; (2) a set of conditions that exist at any given time to define the current state—thus the life cycle observes the conditions associated with particular component instances and determines if it should perform any action; and (3) a mechanism that allows the addition or removal of behavior from an object at run time (giving the effect of supporting "dynamic run-time inheritance"). Although the concepts are explained in the book in a correct and reasonable way, I sometimes had the sensation of being lost in a confusing picture of class hierarchies and mechanisms. Perhaps the wide number of variations and patterns around the state pattern are difficult to understand in a first reading.

A previous reading of the book *SanFrancisco Component Framework: An Introduction* (see the first review in this section) is strongly recommended before starting the advanced topics of this book. After such preparation, the reader will find full coverage of the pattern divided into three sections: an introductory section; a first part describing how the Life Cycle pattern is used and extended by the SanFrancisco order management core business process to build several different order types (sales orders, purchase orders, etc.); and a second part showing how to use life-cycle programming techniques in your own applications. This section details the specific steps that developers must follow when they want to extend the predefined order types of SanFrancisco.

These contents are complemented with concrete application examples, diagrams and code, and some practical tips. The order management domain is used as a basic example to show practical applications of the underlying pattern, but no special knowledge about this topic is required.

After reading through the book, I did not have the impression of discovering novel ideas, rather I found a well-documented set of specific techniques to apply in life-cycle processes of business domains.

If you are searching for technical aspects of the SanFrancisco framework, this book will be suitable for your needs. It is a complete and technical guide to applying this powerful pattern in a wide range of business applications, and to gaining expertise within the SanFrancisco framework.

Mohamed E. Fayad
Computer Science & Engineering
University of Nebraska
Lincoln
Nebraska

*Trademark or registered trademark of International Business Machines Corporation.

Software Project Management: A Unified Framework, Walker Royce, Addison-Wesley Longman, Inc., Reading, MA, 1998. 406 pp. (ISBN 0-201-30958-0).

To begin with, this book is quite readable for someone familiar with software development, but could be considered difficult for a project manager without that familiarity. The vocabulary used is sometimes technical, but that should be acceptable since management of software development projects is a bit different from that of conventional projects. For example, applying project management principles to an iterative development model (like the spiral model) adds considerable complexity to the straight-line project management model.

The author, Walker Royce, developed the approach detailed in this book by managing various software development projects within the government and aerospace and commercial arenas. His project management principles and concepts are not only discussed extensively throughout the book, but are summarized in a detailed case study.

The content of this book (excluding appendices) is organized in the following four parts:

- Software Management Renaissance
- A Software Management Process Framework
- Software Management Disciplines
- Looking Forward

The review that follows covers each of these four sections.

Part 1, Software Management Renaissance, starts by presenting the reader with a good summary of the problems that have persisted in software development since its beginning. A statement made early in this section tends to set the stage, not only for this portion, but for the entire book: "The best thing about software is its flexibility, while the worst thing about software is its flexibility." This characteristic makes the development of software difficult to control, yet is necessary for the creation of software that will have good usability with strong functionality.

This section could be briefly skimmed if the reader has a good knowledge of the history of software development. If not, then it should be read thoroughly, since it sets the stage for the remainder of the book. The author's intent is to make us aware of the many software development problems that have persisted in the past, so we can avoid them through use of a different paradigm in the future.

Part 2, A Software Management Process Framework, attempts to standardize a common process for the development of software. The word "attempts" is used simply because what is presented will be used as a framework from which the reader will need to pick and choose appropriate elements. The framework presented is an architecture that has been segmented into four life-cycle categories: (1) phases (engineering and production), (2) artifacts (management, requirements, design, implementation, and deployment), (3) workflows (activities), and (4) checkpoints (major, minor, and status assessment).

An understanding of the content of these categories is key in choosing which elements to apply to a given project and, consequently, to make the transition from a conventional to an iterative approach to software development and the control of that development.

I have always felt that it was impossible to have a single process for the development of all software. This section of the book supports that premise by emphasizing that an effective project manager must first choose what is necessary and essential for the project being developed, then apply the appropriate development model (i.e., iterative, waterfall, etc.).

Part 3, Software Management Disciplines, takes the process concepts that have been established in Part 2 and applies management discipline to them. First of all, it is recognized that the *planning* of a software development project needs to be iterative, just as the actual development. This is necessary to obtain the correct balance between level of detail planning and buy-in among stakeholders. A rough model of a work breakdown structure is provided as a starting point in obtaining this balance.

Project organizations and responsibilities are discussed, along with the effects (both positive and negative) that a company's organizational structure can have on the makeup of the development team. Understanding these effects, the author defines what he feels are necessary roles and responsibilities, then places them into a project organizational chart.

Even though all the process definition and tailoring discussed earlier are necessary, the fact remains that a significant amount of process automation is also required in order for modern software development projects to operate profitably. Opportunities for automation are identified for each of the work flows previously identified.

Based on the old adage that "you cannot manage what you cannot measure," the following seven core metrics are identified and classified into two categories.

Management metrics:

- Work and progress (work performed over time)
- Budgeted cost and expenditures (cost incurred over time)
- Staffing and team dynamics (personnel changes over time)

Quality metrics:

- Change traffic and stability (change traffic over time)
- Breakage and modularity (average breakage per change over time)
- Rework and adaptability (average rework per change over time)

- Mean time between failures (MTBF) and maturity (defect rate over time)

These metrics are discussed extensively.

Part 4, Looking Forward, attempts to perceive what the future might be in terms of controlling software development. This, along with a discussion of the next generation of software economics, takes us from the metrics that are used today and evolves toward the future by showing how they can be better tuned to the iterative processes discussed earlier.

Emphasis is made on the fact that the management of successful software development projects will continue to be hard work, and there will likely be no breakthrough in the near future. With this in mind, some direction is given to the major cultural shifts that will likely need to occur in the software development community to make the transition from the old to the new feasible.

I recommend the book for reading, with some reservations. Although the book is based on good concepts and principles, much of the information presented may apply well into the future for most organizations. (To apply all the ideas in this book would require the reader's organization to be at a minimum of level three, or possibly four, on the Software Engineering Institute's Capability Maturity Model.) What this really means is that most readers will likely want to apply some of the ideas to their next project and eventually evolve to others as their organizations mature.

Jim Abraham
IBM Learning Services
Rochester
Minnesota

Note—The books reviewed are those the Editor thinks might be of interest to our readers. The reviews express the opinions of the reviewers.