Leveraging Corporate Software Development

Andreas Birrer, Thomas Eggenschwiler

UBILAB, Union Bank of Switzerland
Bahnhofstr. 45, CH-8021 Zurich

e-mail: {birrer, eggenschwiler}@ubilab.ubs.ch

Framework-based development approach – well known from the domain of graphical user interfaces – should be utilized to meet a corporate's increasing demand for customized software. We present the technical aspects of the approach and, based on a scenario of applying framework-based development to the domain of financial instruments trading, we discuss some organizational implications.

1 Introduction

Ever increasing demand for customized software to support specific business processes, especially in the services industry, call for more leverage in the software development process. The leverage must come from two sides. First, a useful approach must yield shorter time-to-delivery and lower per-product-cost than currently employed approaches. Second, the cost of maintaining (making extensions and adaptations) a software system is to be reduced. In our view, framework-based software development is currently the most promising approach in providing such leverage.

In most of today's software development projects much of the same functionality is designed, implemented and debugged for every new application. This is true for structures and functionality in the user interface, the data storage subsystems, the communication facilities, etc. But, it is also true for structures and functionality that appear very closely linked with the specific solution a certain application provides.

This generally observed lack of reuse-orientation has two reasons. First, the design scope and the implementation scope is typically one project. The result of this setting is that there is neither time nor incentive to redesign a working solution. The second reason is the lack of a vehicle to capture concepts and software solutions such that they are attractive to reuse.

There exist different notions of what to call a framework. When we talk about framework-based development, we mean the kind of frameworks that are supported by object-technology [Joh88].

This paper is organized as follows. Section 2 presents the framework idea and the technical aspects of framework-based software development. In section 3, we lead through a scenario showing how a financial institution might tackle its software needs for supporting financial instruments trading. Simply adopting the techniques and technical guidelines associated with framework-based development will not make use of the full potential of the approach. What is needed are radical changes in the way software development is organized, managed and valued. This is the issue of Section 4. Section 5 summarizes the cornerstones of framework-base development.

2 Technical Aspects of Frameworks

An object-oriented system is a society of objects. A designer distributes required functionality among cooperating objects. If a task is sufficiently complex it is implemented by several objects working together as a team. In such a system, each member has its assigned role.

Objects collaborate by sending messages to one another. The set of messages that an object understands is called its protocol. For example, a user interface object might ask a password broker object whether the character sequence the user just typed in is a valid password. The password broker object may not have enough data on its own to answer this request, but it knows of a few password database objects. One of them will certainly have an answer.

2.1 Frameworks

Every software system has functionality that is specific with respect to the problem it tries to solve. However, looking at several applications at once we also find that they have some functionality (or set of roles) in common. This recurring functionality is termed the *generic part* of an application. The functionality needed to provide one particular solution is termed the *specific part* of an application.

A framework implements a generic part and defines the places where extensions have to be made in order to support specific solutions. Object technology provides constructs and the techniques to support this separation of the generic and the specific part of an application directly on the code level. The extensions can be quite small in comparison to the rest of the code.

Now, returning to the password broker example above. Let us assume several applications need password management but they have slightly different requirement on how passwords are check. A framework might then supply a password broker object as one element of its generic part. Other objects in the system whether they belong to the generic or the specific part can rely on the existence of this password broker object. To account for the differing requirements, the password broker object can be configured with a security strategy object which encapsulates, among other things, how passwords are checked. Such a security strategy object belongs to the specific part of an application. The framework defines what messages such a security strategy object must understand.

In a rich framework much of the functionality to provide the required behavior of the application to be built is already present. But at various places this generic functionality depends on functionality which can only be implemented with respect to one specific application. The protocols (set of messages) required of objects of the specific part insulate the specific part of the system from the framework and make the latter independent of the specific objects. Figure 1 illustrates this schematically and compares framework-based with “function library based” development.

To summarize, the flexibility and genericness of a framework stems from only relying on what is done (and where), and abstracting from how it is done. Object technology allows to carry this decomposition of responsibility much further than previous technologies.

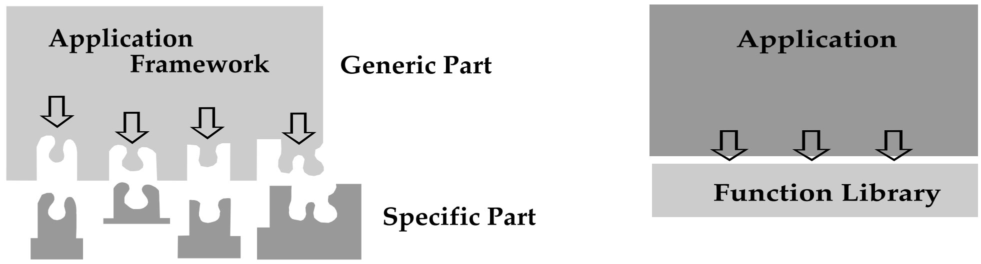

Figure 1. Framework-based vs. function library based development.

2.2 Patterns

A rich and powerful framework may become quite complex and it is difficult for a non-expert application developer to grasp the “big picture”. This may result in a serious entry barrier. Fortunately, the object-oriented community is increasingly aware of certain recurring designs that lead the developer to recognize and apply common solutions to the problems they try to solve. These designs are term patterns [Gam93, Cop94]. To the developer who is aware of these patterns they work like filters through which the complexity of a framework is greatly reduced.

2.3 The Scope of a Framework

Prominent frameworks are mostly associated with the domain of graphical user interfaces [Wil90,Wei89]. But examples in other domains exist as well (e.g., operating systems [Cam91], distributed applications [Sch94]).

To provide maximum support in application development the scope of a framework should be confined to a rather narrow domain. That is to say, a framework should mainly support a certain breed of applications. Such a selection of applications is sometimes called a *product family*. In the following section we develop a scenario which accounts for the need to focus on a product family when developing a framework.

3 A Scenario of Framework-based Software Development

In the following scenario we look at a financial institution starting to offer financial instruments over-the-counter. For sake of brevity, call it a bank. To support its business this bank, among other things, needs adequate software. It decides to develop this software itself using object technology, and thereby build up the necessary know-how in-house.

In designing the software, one of the biggest problems faced with is that the evolution of the business and the range and depth of trading activities of the bank can hardly be foreseen. Emerging new opportunities can have drastic impact on the breadth of the services offered. The software system must keep up with new activities of the trading teams and support new services. There is no way to perform an exhaustive analysis and come up with one optimal system. Also putting to work a certain solution will almost certainly trigger new requirements.

3.1 Implementing a First System

The bank first wants to offer FRAs and interest rate swaps tailored to the needs of its customers. For this it needs software for pricing individual FRAs and swaps, and access to real-time market data. A back-office application supports deal execution and payment settlements of active contracts. To perform risk management traders and senior management need profit/loss figures and exposure indication (interest rates, counter parties). A common data base integrates the different applications.

3.2 Extending Trading Activities

After some time, an other trading group is assigned to trading in options. This will also include products like caps, floors, and swaptions.

The system requirements for the options trading team are much the same as for the FRA/swaps team. The main difference lay in the pricing software and the risk management software. In addition, the data model has to be extended.

Instead of building a new system from scratch, the development team decides to follow a reuse-oriented approach and looks at the software for swaps and FRA pricing to find ways of extending it. Equipped with the knowledge of how swaps pricing software works and the new pricing requirements the developers build a pilot application for the pricing in the option domain. During this development it becomes clear that both applications will have much in common despite the fact that the pricing models used are quite different. In the quest of an elegant solution and with maintenance in mind the developers take a step back and rework their design of the option pricing software. The more generic algorithms and attributes thus found promise to work well in the context of the instruments that must be supported so far. After implementing the option pricing software the developers decide to reengineer the swaps pricing software based on the new insight and the generic software components.

3.3 Including Portfolio Management Services

When business activities are extended by including portfolio management as a customer service, it is recognized that it would take a system similar to the one in the trading department to support the new services. Since the bank is aiming at a corporate software base the best move is to assign as many of the trading system developers as possible to the task of building a portfolio management system.

The framework that exists so far provides leverage in building pricing and risk management in trading software. So this framework is taken as a starting point to implement portfolio valuation and various models for portfolio performance. During design and prototyping of the portfolio management system the developers recognize various new ways of implementing instrument definition and valuation, and refine the framework to incorporate these insights.

To further support development, a corporate software base must provide software components that cover inter-application communication, integration of real-time market information, and transparent access to data in various databases, even if data models evolve over time. Such support is increasingly supplied by commercial products and need not be developed and maintained in-house. What is needed, however, are vendor independent interfaces to such components. These have to be developed in-house.

The know-how gained in past development must be secured. Considerable know-how recovery was already performed during redesign of the valuation and risk management framework. But that framework will need extensions in order to stay alive. Additional corporate wide software support such as vendor independent interfaces need to be included as well. The bank’s choice is to assign the task of maintaining this corporate software base to a framework group. This group includes some of the developers of the trading department software. Given the implementation leverage that the framework group provides then small teams can be assigned quite formidable development tasks. As part of maintenance, older applications are reengineered to the extent that they comply with the standards defined by the framework.

3.4 Offering a Range of Highly Customized Financial Products

While the spreads (and hence profit) in trading relatively simple instruments is continuously shrinking the business of providing custom-tailored financial instruments gains on appeal.

The valuation of a wide range of tailor-made financial instruments requires that the calculations and the logic in supporting software provides appropriate flexibility. This flexibility is only achieved if valuation is based on building blocks rather than entire instruments [Smi87, Err94].

When the developers look at ways to support customization they soon recognize that much of the required flexibility is already available in the framework. The generic software structures that until now served implementation must only be made explicit.

Finding further building blocks and ways of composition is as much an evolutionary process as the refinement of the framework to incorporate option valuation and portfolio management support. In a number of successive developments the underlying concepts of the valuation of instruments are identified and modeled in software. Once made available by the framework in software form they can be reused in other developments while the search for other concepts and more powerful building blocks continues. As one such development, the portfolio management software can be extended to include valuation of more complex instruments thus giving the portfolio manager a broader choice in strategies.

4 Framework-based Development Organization

When a corporate adopts framework-based development it must deal with a number of challenges:

- New requirements and design solutions must find their way to the framework developers.
- Frameworks undergo consolidation and redesign periodically. The changes must flow back into applications that are still being maintained and extended.
- Generally, programmers new to the framework must be informed about and sensitized to the abstractions and patterns available in the framework. This is extremely important since otherwise they will reinvent solutions, and the benefits of reuse are lost.

These are not really technical challenges. Rather they must be dealt with organizationally. In the following, we outline an organizational structure that addresses the special characteristics of framework-based software development.

4.1 Separation of Framework and Application Development

When developing applications based on a corporate framework then software projects should be organized around a central group which is concerned with the development and the maintenance of this framework. We call this central group the *framework maintenance group* (FMG). The FMG is a sort of competence center for framework and concept development.

Why this separation? The answer is one of application scope. While a good developer may discover and apply powerful abstractions during application development these abstractions server primarily the implementation of the required functionality. They don't automatically fit the requirements of other projects because the separation of generic and specific functionality is with respect to one set of requirements. Only with respect to the requirements of several application can anyone decide what to consider generic and what to consider specific. So, what it takes is a periodic redesign of already working software through refactoring. A mature framework has undergone several design iterations and thus aggregates the know-how gained from several application developments.

The pressures from release plans and the resulting short term view discourage an application developer from reworking the design. Thus refactoring does not happen spontaneously and must therefore be institutionalized.

4.2 The Framework Maintenance Group

The main function of the FMG is to look at software solutions for a class of problems from application development and try to come up with a generic solution. So the major input to the work of the FMG are the design solutions arrived at in past projects. The FMG is staffed with developers who have the necessary skills and motivation to grasp generally useful abstractions and evolve them through successive redesigns into generic solutions.

Another important function of the FMG is to relief the application implementor from system dependencies. The application implementor is offered hardware and software system independent interfaces to system resources (operating system, window system, communication, data bases, etc.). Members of the FMG specialize in state-of-the-art abstractions to system resources. The result of centralizing this function with the FMG is the development of an in-house standard for accessing system resources.

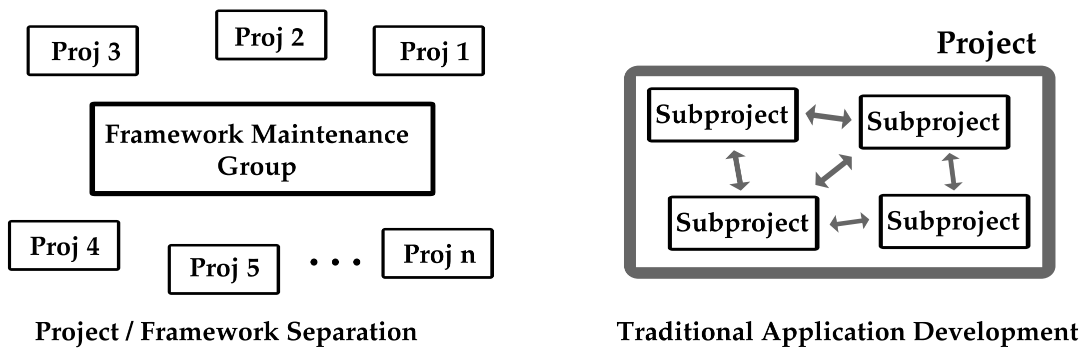

Figure 2. Comparing Development Organizations

4.3 The Effect on Project Team Size

In traditional development without reuse-orientation, the range of functionality to be implemented of even a medium sized application requires that development is split up in several subprojects. Since the entire infrastructure and the architecture of the application needs to be engineered there is considerable communication necessary to synchronize the development efforts. This is shown schematically in figure 2 on the right.

When development is based on a suitable framework then most of the infrastructure and the overall architecture is reused. The developers can concentrate on the implementation of the specific functionality. We can therefore assign a much smaller number of developers to a project. Few project would require more than three developers.

4.4 The Role of the Mediators

The separation of the developers in the FMG and the application developers introduces a new problem. As mentioned above, the FMG must be kept informed about the problems that application developers are confronted with. Application developers find new solutions when designing the specific part of an applications. These solutions are very valuable to the whole software process and must flow to the FMG for integration in the framework. Changes and extensions to the framework must propagate to those using the framework.

One way to attack this problem is by job rotation. After having completed a project an application developer spends a few months in the FMG and then moves on to an new project. These mobile developers effectively act as mediators between the FMG and application teams. The personal relations established during job rotation promotes know-how exchange long after.

Not every developer is suited to take on the role of a mediator. Thus application design reviews with one or more FMG developer participating supplement the role of mediators in know-how transfer.

A very important aspect of mediators is the promotion of a *framework culture*. This is the collective understanding that reuse of existing solutions and suggestions for improvements of the framework are vital for successful development of the corporate software base.

5 Conclusions

Software development today is characterized by applications that require a wide range of functionality (graphical user interface, calculations, database access, inter-application communication, etc.). The business activities which these applications support keep changing as market places evolve.

A mature framework provides considerable leverage for corporate software development through major reductions in:

- development time
- team size
- development risk
- maintenance cost and time

Frameworks are repositories of both concepts and real software solutions. This property of frameworks helps to secure the often large investments into new designs and the acquisition of implementation know-how. The very process of framework development transforms the implicit know-how accumulated in the developers during problem solving into something explicit and reusable. This know-how is thus protected from loss due to employee fluctuations.

Frameworks are generally difficult to develop. It takes experienced architects with a passion for reusable software structures. The wider the domain to be supported the more the requirements of different application will diverge and the more difficult it becomes to find abstractions that work well for all applications. A good and reusable distribution of responsibility is typically not evident at the outset of development and requires a creative process with successive redesigns. The software development organization must institutionalize consolidation of design across projects and foster a reuse culture by rewarding reuse.

We recognize that the approach presented in this paper is not complete. We have not talked about how people and teams influence the successes or failures in software development. The whole question of why one project is a success while another project having access to the same technology fails to yield any results would be outside the scope of this paper. However, a wealth of insight on this topic can be found in [DeM87].

6 References

[Cam91] Campbell R H, Islam N, Johnson R E, Kougiouris P, Madany P: Choices, Frameworks and Refinement. In *Proceedings of the International Workshop on Object Orientation in Operating Systems* (Palo Alto, CA, October 17-18), IEEE Computer Society Press, Los Alamitos, CA, October, 1991

[Cop94] Coplien J O: Software Design – The Emerging Patterns. In *C++ Report*, Vol 6/No 6, July-August 1994

[DeM87] DeMarco T, Lister T: *Peopleware – Productive Projects and Teams*. Dorset House Publishing Co, New York, 1987

[Err94] Errington C, *Financial Engineering – A handbook for managing the risk-reward relationship*. MacMillan Publishers Ltd, London, 1993

[Ferr89] Ferrel P J, Meyer R F: Vamp – The Aldus Application Framework. In *Conference Proceedings of OOPSLA'89*, October 1-6, 1989

[Gam93] Gamma E, Helm R, Johnson R E, Vlissides J: Design Patterns: Abstraction and Reuse of Object-Oriented Designs. In *Proceedings of ECOOP'93*, Springer-Verlag, 1993

[Joh88] Johnson R E, Foote B: Designing Reusable Classes. In *The Journal Of Object-Oriented Programming*, Vol 1, No 2, 1988, pp. 22-35

[Joh93] Johnson R, Palaniappan M: MetaFlex – A Flexible Metaclass Generator. In *Proceedings of ECOOP'93*, Springer-Verlag, 1993

[Sch94] Schmidt D C: ASX – An Object-Oriented Framework for developing distributed applications. In *Proceedings of the 6th USENIX C++ Technical Conference*, Cambridge, MA, USENIX, April 1994

[Smi87] Smithson C W: A LEGO Approach to Financial Engineering: An Introduction to Forwards, Futures, Swaps and Options. In *Midland Corporate Financial Journal*, Winter 1987, pp. 64-86

[Wei89] Weinand A, Gamma E, Marty R: Design and Implementation of ET++, a Seamless Object–Oriented Application Framework. In *Structured Programming*, Vol 10, No 2, 1989, pp. 63-87

[Wil90] Wilson D A, Rosenstein L S, Shafer D: *Programming with MacApp.* Addison-Wesley, Reading, MA, 1990