ImageBuilder Software

A Framework Development Experience Report

Dwayne Towell
ImageBuilder Software
6650 SW Redwood Lane, Suite 200
Portland, OR 97224
dwayne@imagebuilder.com

Key words: Framework, experience, object-oriented, cross-platform, multimedia, reuse, education, domain, team, architects, development

Abstract: Six years ago ImageBuilder Software chose to develop an object-oriented, cross-platform, multimedia framework to promote code reuse and therefore increase profits. Today, it continues to be used and extended; it is profitable and a major company asset. This paper documents how it was developed, describes how we use it today, evaluates its success, and makes recommendations for others based on our experience.

1. THE COMPANY

ImageBuilder is an independent, full-service, multimedia title development company. Founded 15 years ago, it now has over 120 full-time employees including more than 30 engineers. We design, develop and test CD-ROM titles for clients, partners and, more recently, our subsidiary Active Arts. Many products are completed independently from conception through development to manufacturing release by ImageBuilder staff, however we do allow clients to participate in the process to the extent they desire. Most of the dozen or so products shipped each year are dual Windows and Macintosh, shrink-wrap, edutainment, multimedia CD-ROMs. Some of the most well-known titles include: Hasbro's *Pictionary, Mr. PotatoHead*, and *Playskool Puzzles*; Creative Wonder's *Madeline Classroom Companion* series; The Learning Company's *Math Munchers*

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35563-4_35

P. Donohoe (ed.), *Software Architecture*

Deluxe and *Paint, Write, and Play*; Pacific Interactive's *Bill Nye: Stop the Rock!*; Disney's *Disney Magic Artist*; and Microsoft's *Arthur's Playground.*

2. PROJECTS

Each project is somewhat unique, however most follow a well-traveled path. ImageBuilder producers collaborate with the client to develop an outline for the product including scope, content and purpose. As this is nearing completion, the project lead and art director are assigned to begin developing a product specification. This is a working document so changes are quite regular especially near the start of project as details are ironed out with the client, engineers and artists.

As areas of the product specification become firm, engineers start on the technical specification. Engineers outside the project will usually critique it in one or more design review meetings. Once the product specification is complete, the Quality Assurance department will develop plans for testing and certification. Additional members are added to the project as they are needed and/or become available. Eventually, the team will include two to six engineers, one or more media coordinators, artists, animators, scriptwriters and/or sound designers. Most projects employ about eight full-time positions and tend to last about ten months, but they can vary quite a bit. At any time, ImageBuilder has a dozen or more projects in progress.

3. SUPPORT

One important ingredient contributing to a successful project at ImageBuilder is our proprietary, object-oriented, multimedia framework and its associated tools. Framework code comprises from one third to one half of most applications developed. Typical applications make use of about 80% of the code provided. Since practically all development builds on our framework, a framework development team continues to improve it and provide support for its use. The framework team's initial responsibility is to develop, improve and extend our domain-specific object model for multimedia applications. This model is realized as an object-oriented C++ framework. Several tools have also been developed to allow the large quantities of multimedia resources to be manipulated, compiled, and viewed.

In addition to developing the object model, delivering code and providing tools, the team provides design review and education. Most projects take advantage of the team's design experience during formal reviews. Also, engineers frequently use the team members as a convenient consultant for

object design. Education provided by the team plays an important part in the improvement of the engineering department. Whenever consulted the team tries to take advantage of these "teachable" moments. In addition, formal classes are held weekly covering various topics from engineering processes to use of the framework to object modeling.

4. ARTIFACTS

The framework team maintains three artifacts: C++ framework, documentation and tools. The framework is actually multi-tiered; it includes an operating system services (OS) layer, an application layer and the multimedia framework proper. The OS layer models services such as files and threads, providing cross-platform objects to hide platform-specific details. The application layer builds on the OS layer to provide windows, controls, menus, events, etc., again hiding platform-specific details as much as possible. Finally, the framework proper models multimedia objects such as pictures, sounds, animations and buttons.

Table 1 shows the number of modeling objects and additional utility classes for each layer.

Table 1. Domain Objects by Framework Layer

	Domain Objects	Other Classes	Total
OS services	26	53	79
Application framework	20	27	47
Multimedia framework	88	118	206
Total	134	198	332

In addition to framework code, extensive documentation has been developed. Approximately 60 pages of overview and intermediate-level documentation are currently available on an internal web site. This continues to grow as the framework is extended or new areas are identified for further explanation. Class-level documentation, aimed at application use, is provided with the class in header files.

The framework team also has responsibility for several tools including a content compiler, a script compiler, an animation viewer, a resource browser as well as others. These tools allow resources to be converted, compiled and viewed for shipping and run-time use. Resources supported include: text, pictures, sounds, MIDI, run-time composed and streamed animations, QuickTime movies, Windows AVIs and project-dependent extensions.

5. HISTORY AND EVALUATION

Seven years ago ImageBuilder began to respond to industry changes as Windows 3.0 and 3.1 were shipped. Emphasis changed from business graphics to multimedia products. Six years ago we started our third multimedia project and decided much of each project could be reused if we could developed a "multimedia engine". So a parallel project to create reusable code was started in conjunction with the client's project. Although extremely primitive by today's standard, it gave us our first chance. The most important thing we learned was the need for extension—the need for a framework, not an "engine". It also allowed us to "throw one away".

At the conclusion of that project we immediately went back to the drawing board. We allocated two of our best engineers and completely redesigned the object model from the ground up. Over the next year or two we continued to improve and extend the model. Several projects were now using it and a few had even been completed. By the end of this period we shipped about 6 products using the framework, but had also come to realize many of its shortcomings.

The third year into the project we went back to the drawing board again. Although we eventually touched all the code, much of the domain-related object model remained unchanged. Following this redesign was an extensive period of conversion. The framework team was under pressure to deliver vast amounts of new code for projects underway. In response we tried using occasional part-time team members either to help with conversion or develop new areas. In the end, this did not work because our part-time engineers did not have experience developing extensible code.

As the percentage of projects using the framework increased to almost 100% and the engineering department grew, we had to improve our processes. We added an administrative assistant and an official release procedure. Releases now included detailed change reports, verified code for all supported platforms, and tools synchronized with the run-time code.

Over the years many additions have been suggested for the framework. We have never lacked for proposed improvements and as each project pushes the envelope, pressure to make enhancements increases. The framework team periodically reviews the framework with two groups. Producers are consulted for strategic direction. Engineers are consulted about utility and convenience. Based on these directions the framework team develops short- and medium-term goals.

In addition to application team needs, framework additions must meet two requirements. First, the addition should be useful to more than one project since the framework exists to reduce costs by increasing code reuse.

Second, the addition must be well defined. Open-ended additions become sinkholes for time and effort.

The need for education was one of the slightly unexpected results of developing a framework. Although conversant in the C++ language, many of our engineers lacked object skills. In an attempt to improve skills and deliver timely information about the framework, weekly engineering meetings were established. Time is divided between teaching object skills, improving engineering practices, and discussing framework use.

The framework has been a major benefit to project development at ImageBuilder. It allows application teams to do work faster because many of the details are already solved. For example, rapidly prototyping a game, module or entire application can be done as fast as content becomes available. Also, it improves the quality and speed of applications ImageBuilder is able to ship. Each product that takes advantage of the framework contains proven and optimized code for its core functionality.

6. RECOMMENDATIONS

6.1 Getting Started

Building the framework team is the first step to a successful framework development. The team will be responsible for deriving the object model, documenting it for project engineers, implementing it in code, and supporting it.

6.1.1 Create a Framework Development Team

You will not have well-architected, reusable code if no one has primary responsibility to develop it. As much as it would be nice to believe good engineers would develop reusable objects and code, it is very difficult while under pressure to meet deadlines. Our experience indicates it will never happen. However, it can be accomplished given a team whose full-time responsibility is framework extraction or invention.

6.1.2 Enroll the Best Architects

Design is first chance engineers have to influence a project for success. Correct decisions pay off for the rest of the project, while mistakes made here cost the most to correct. Given a good domain-specific framework, much of the design for a project has already been done. So, to make the most of your investment, allocate the best architects available to framework

development. This allows all projects, and engineers, to benefit from their experience and knowledge.

6.1.3 Allocate Ten Percent

Allocate enough engineers to be productive and make a real contribution, but not too many to be unmanageable or risk extensive overhead. Our team has varied slightly over the years, but we have found allocating ten percent of our engineers to be about right.

6.1.4 Promote Stable Membership

Team membership should be stable. Framework development is fundamentally different from application development. For example, correct framework design is typically more important than the schedule. Also, inventing a quality model is considerably different from using it. Therefore, enroll new members for long terms, a year or more, and make key members permanent. Since members will be working together for extended periods, and in some cases indefinitely, chose the team carefully and use trial periods. This allows everyone to reevaluate the assignment after an agreed interval. If the situation does not appear to be working, the trial can be terminated with less discomfort for everyone involved.

6.1.5 Empower a Visionary Leader

As with all enterprises, a strong leader is needed. Someone who has a clear vision for the framework needs to "own" the project. Especially before the project is well established, but even later, it will be pulled in many directions. Each client project will make a case that the framework team should solve its special requirements. An empowered leader will be able to hold it on course and allow it to meet the widest possible needs.

6.2 Making Progress

Once the project gets underway the team will be developing the domain object model. It is unlikely any team will "get it right" the first time around. Even if the first version is successful, plan to improve and extend it.

6.2.1 Make It Tractable

Only consider taking on manageable areas of the domain. Some areas will be complex or ill defined; ignore them. In many cases the application

engineers will be able to solve the subset of the problem needed for their application without needing the general solution. Even though a general solution may be enticing, especially to architects, consider the economics of building it.

6.2.2 Make Official Releases

As with any construction project, building on an unstable base is tricky at best. Give the application engineers a hand by producing "official" code releases. Produce "release notes" that announce model changes, extensions and bug fixes. Typically project engineers are much happier to receive new code if they know what has changed. Balance the desire to release changes to application engineers with the cost to perform a release. We make releases at most once a week, but they may occur less frequently if few changes have occurred.

Suggest each project archive its own copy of the framework. This allows an application member to control migration to new releases when convenient for that project. It also allows application engineers to make local changes and bug fixes after framework development has been "frozen" for that project.

6.2.3 Keep the Model Stable

Backward compatibility is important with rapidly changing code. For frameworks this means the domain model must support the ways the application engineers use it. Application engineers hate releases with architectural changes. Although improving the model may require "code breaking" changes, try to keep them to a minimum. Find temporary ways to support "old style" objects to give engineers time to convert.

6.3 Developer Relations

In addition to developing a model and providing code for it, the framework team will spend a considerable amount of time supporting their customers, the application engineers.

6.3.1 Solicit Client Input

The framework team will increasingly lose touch with application development as it concentrates on the framework. Therefore, develop an ongoing dialog with framework clients (i.e., the project engineers). Goals for

the framework will change over time. Some method of constantly reevaluating them should be available.

6.3.2 Promote Object-Oriented Skills

Even if the quality of the object model developed is excellent, application engineers will need object-oriented skills. In order to insure the success of the framework it may be appropriate to include object education as framework support.

6.3.3 Assist in Using the Framework Effectively

Even if application engineers have object-oriented experience, solutions to some problems may not be obvious. During framework development many domain problems will be considered and plans made to allow for their solution. If application engineers are not aware of these proposed solutions they may not use the framework effectively. Since the framework team will be the experts within the domain, take advantage of design reviews to assist application engineers with domain-related object design.

6.3.4 Refuse Ownership of Project-Specific Problems

When assisting project engineers, pressure will mount to use framework team members to solve project-specific problems. While this is most evident near the beginning of framework development, it will continue to plague the team years later. Resist the urge to allow framework engineers to participate on an application team. Framework members should be consulted on issues of the framework or for advice but application development should remain separate.

7. CONCLUSION

Developing a domain-specific, object-oriented framework has allowed ImageBuilder Software to remain competitive, grow and succeed in the fast-paced environment of software development. Providing applications within the framework domain has proven reliable and profitable over the past six years. While most of the techniques presented have a proven track record, some of the ideas only become clear in hindsight. Though ImageBuilder developed these recommendations through trial and error, we count the project as a success. Other organizations, with the benefit of these recommendations, should experience smoother sailing.

ACKNOWLEDGEMENTS

This paper was prepared for presentation at WICSA1. Thanks are due to Roger Bonzer for his many comments and Terry Hamm for providing the opportunity to write.