My First Jasmine Project

By Bill Cross

CA-World 2000
eBusiness Solutions in Internet Time
JG065SN

Introduction

This paper presents an honest look at the successes and lessons learned resulting from the development process of our first Jasmine application. This paper will review issues and decisions regarding scheduling, cost estimation, User Interface (UI) language choices, development tools, data model development, ODQL issues and other relevant topics. The intent is to provide an opportunity for developers and managers to learn from our experience. Any comments or questions are welcome and may be addressed to the author at BillCross@vwi.com.

Project Overview

The subject project was to develop a prototype supporting the management of information gathered during the Army medical master planning process. This process involves the complete review of a regionalized health care system supporting United States Military members, retirees, their collective dependents, and other authorized beneficiaries. This review focuses on all information that is deemed pertinent to real estate capital improvement and repair decisions.

The basic problem being defined and resolved is whether a particular medical facility and its network of clinics are the best candidate for limited facility repair, renewal or replacement funds. In order to answer this question, each facility is analyzed from multiple perspectives: the building and its physical condition; how well it supports the current and proposed future operations; the efficiency of the current operations (correct staffing numbers); projected changes to the future beneficiary (customer) populations and the medical services they demand; and finally the integration of any decisions into a fiscal budget spanning one to five years.

Types of information being gathered included:

- Architectural drawings representing the buildings' current layouts and also proposed alternatives for future "optimized" layouts.

- Condition assessment documents and reports assessing each of the buildings' engineering systems, e.g. electrical, heating, ventilation and air conditioning (HVAC), medical gases or transportation. These assessments would often include substantial narratives, exemplary photographs and subjective (but scaled) ratings as to the conditions.

- Tabulations of each space within the building as it is currently being used and also of the rooms that are proposed for the future.

- CAD templates used as typical room layouts and representative of the desired “Criteria” standard to be attained.

- Detailed records of historical patient visits providing details of patient demographics in correlation with medical services being requested

- Records of cost accounting attributed to the various medical services.

- Population projections for the local region around each medical facility.

The initial goal of this project was to create a database application that would support the basic storage of all the above output from the master planning process. The ultimate goal was for the application to be capable of evolving to become part of the master planning process. It would ultimately enable the analysis of the above disparate sets of information across regions, medical services, and medical networks. And, to do all of this on a shoestring budget.

Beginning

The project began following a discussion between the client and our senior management. As a result of this meeting a funding scope was discussed. The initial desired estimate was $125,000, based primarily upon the availability of funding and not the result of analyzing the requirements. The latter would have been difficult in any case, since the requirements were very loosely phrased. The stated intention was for the project to allow electronically storing the results of the master planning process. The client requested to have the product within a year. Fortunately for me, I was requested to personally lead this effort since I had recently left the client’s organization. It was felt that I understood the client’s needs better than they did. I made the mistake of believing this too.

After discussions with the user and my boss, I attempted to write a statement of scope that would match what I understood to be the user’s desired result. I had developed applications previously, but always functioning as both user and developer. This was to be my first Jasmine application, and due to the request for completion within a year, I would also need to partner with other developers to accomplish the project. My company had the added requirement to structure this contract as a fixed price type of proposal. This might work reasonably well where the requirements and functionality had been clearly defined; however, those were part of the initial deliverables for this scope.

Project success normally stands upon a three-legged stool of quality, cost and time. As the result of the initial discussions with the client, two of the legs had already been established. I adjusted the scope statement to stipulate that this effort would result in a “prototypical” product. I also continued to define the scope to present a product that would meet what I believed were their realistic needs. They needed a database product that would support the various interrelated items of information necessary to properly determine and allocate funding for real estate development. This database needed to be capable of storing various types of data, and be scalable to meet the database's future growth.

The ability to adapt to not only the changing real estate solutions, but also to changing solution processes was an essential requirement to long term viability of the project solution. Jasmine through its object orientation met these needs handily, since it supported all types of media, was scalable, relatively inexpensive (when compared against and Informix or comparable installation), and through its pure-object implementation provided an "…enabling technology for adaptive business systems."[1]

Lessons Learned

Following are some of the main lessons learned from this, my first Jasmine project.

Development

Team

Immediately upon receipt of the contract, we established a team consisting of myself as project manager, functional analyst, object modeler, and technical writer. Another object modeler was added as part of the team, and coding was performed through use of a subcontractor. Their staffing grew from one programmer in the initial months to six programmers toward the end of the effort. This type of staff growth is to be expected, since the construction phase is expected to entail 65% of the effort[2]. Having two data modelers was the right number, however, I definitely played too many roles in the project.

Design and Approach

The decision regarding the back-end data server was made at the project's very beginning to be Jasmine. However, the decision as to the best GUI language was still to be determined when the project got under way. We considered four development platforms:

1. Jasmine Studio v1.2
2. C++
3. CA-Visual Objects (v2.0)
4. Java with Java Proxies (on the Jasmine installation CD)

We had three main considerations for each platform. The ability of the language to work with Jasmine, the language's internet support capability, and the availability of programmers knowledgeable in the selected language. We felt that while the Jasmine Studio v1.2 software worked very well with Jasmine and did support the Internet well, it had two serious limitations. First, the availability of knowledgeable programmers was extremely limited at the time we were considering it (mid 1998). Second, the design interface did not allow the programmer to work with script. Instead the programmer was forced to work solely through the design interface. We felt that this would limit the speed at which the programmer could develop, since all actions had to be accomplished through the development GUI. Cut and paste operations while working in code were not an option. We also did not have a budget that would allow learning a new tool.

C++ was considered but was felt to be too difficult to work with for a prototype. The Jasmine C-API provided complete access to Jasmine's functionality, but C++ as a tool did not come with a framework for the application.

[1] Taylor, David A., 1998. Object Technology A Manager's Guide, 2nd Edition, p1

[2] Kruchten, Philippe, 1998. The Rational Unified Process, An Introduction. p118.

CA-Visual Objects version 2.0 was also considered. The other modeler and myself were both VO programmers and would have liked very much to develop the GUI in this our favorite language. However, while v2.5 promised to provide strong Jasmine functionality (and in truth v2.5 does indeed make working with Jasmine relatively straightforward), at the time when we had to decide, v2.5 was still a future product with an uncertain release date.

In consideration of the above, JAVA was chosen as the GUI development language. It provided a reasonably strong interface with Jasmine. It was also a language with a large and growing number of programmers. However, I found that Java like C++ did not come with a ready-made database framework. The basic requirement to rapidly build menus, create data access windows with database navigational abilities all had to be manually developed.

If the decision had to be made again today, I would definitely select CA-Visual Objects v2.5 as the development language. With its strong interface to Jasmine through its own proxy objects, plus its ability to call Jasmine commands directly it has all the capability of Java with Java proxies. However, it also has a ready-made framework for database application development, with the ability to rapidly create a basic application with menus and navigational data entry screens immediately available out of the box, while also allowing complete program control through compiled script and a C API. It also has a strong speed advantage when running in a Windows environment, which was the client's principal operating system.

The design approach was to maximize the use of Jasmine to store all aspects of the basic business of master planning. We wanted to fully implement the object concept through Jasmine. Our business objects would contain the business data in the form of attributes and also the business logic in the form of methods. A typical application has most of the business logic written into the source code of the GUI application itself, with only the data being stored in the database. We wanted the Graphical User Interface (GUI) to server purely as the user interface with minimal business logic.

This separation of GUI and server came about from a few reasons. Although the initial project prototype was for a network-based application, it was believed that the production application would demand an Intranet solution. Again Jasmine as a backend supported both readily, the issue was making the GUI a powerful interface in both environments. The Java (or C++ or VO) decision was supportable since all users of the future application would be part of the company and would be able to download the necessary software. The delivery model was closer to that of AOL or CompuServe, where the interface application must be mailed to the customer on a CD, as opposed to a pure web HTML interface and delivery scheme.

Our intent created interesting challenges in trying to keep the GUI dumb. Even things like the construction of tree views for exploring through data was determined to be something that should be provided by the Jasmine database. It was decided that each class should be able to tell the GUI how the hierarchy of its attributes should be grouped together to create an explorer tree view. To require the interface to sort through various attributes and layers of contained objects would mean that the GUI was aware of the basic business rules of ownership and relationships among the various objects. A modeling approach to solving this resulted in the design of intermediate classes. The GUI would make a request to an intermediate object for specific information such as a chart. Information requests would be determined by the methods available on the chart intermediate object. The chart intermediate object would then be responsible for calling to the respective business object asking it to populate the chart object with data. When the object was populated it was then passed back to the GUI which would be responsible for format and presentation. The creation of such intermediate classes was readily supported through Jasmine and the Java proxy methodology.

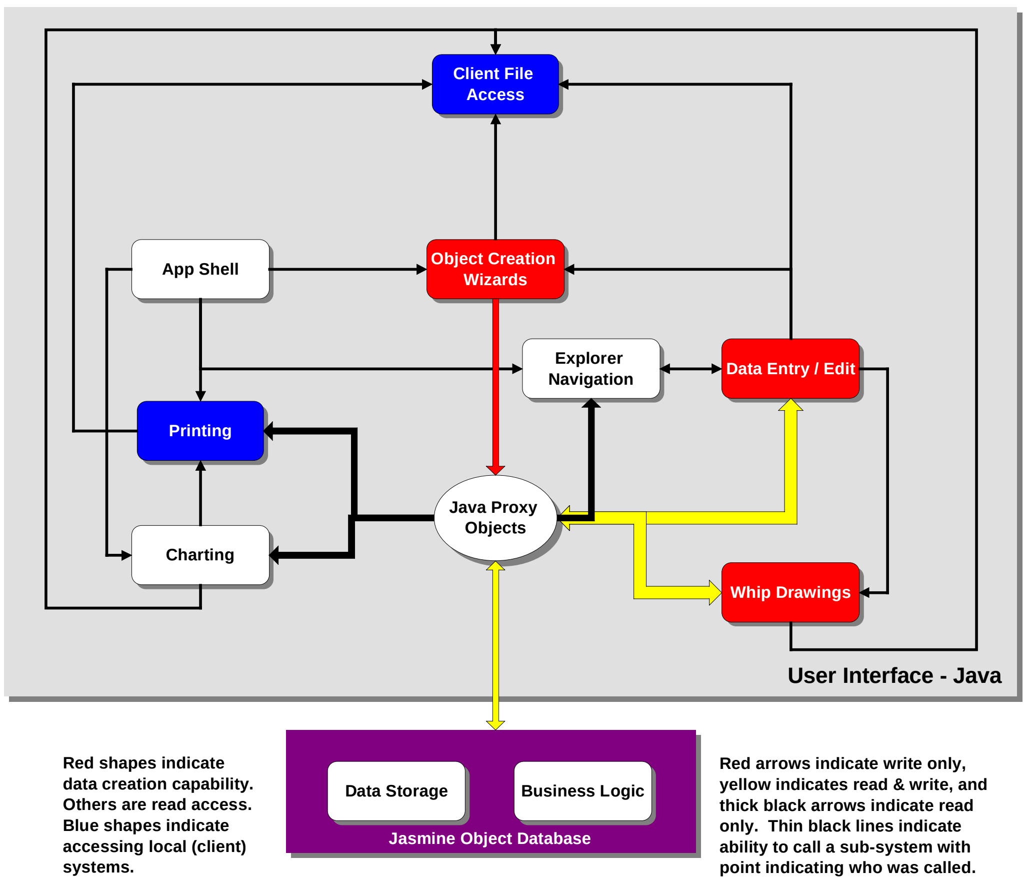

Client File Access
App Shell
Object Creation Wizards
Explorer Navigation
Data Entry / Edit
Printing
Java Proxy Objects
Charting
Whip Drawings
User Interface - Java
Data Storage
Business Logic
Jasmine Object Database
Red shapes indicate data creation capability. Others are read access. Blue shapes indicate accessing local (client) systems.
Red arrows indicate write only, yellow indicates read & write, and thick black arrows indicate read only. Thin black lines indicate ability to call a sub-system with point indicating who was called.

Figure 1: Diagram of resultant split between GUI and Server

Modeling

A large amount of the effort both in terms of schedule and man-hours was devoted to modeling the problem and its solution. Performing this modeling required a tool that would allow me to share the results with the rest of the team. Ideally, each member of the team would have a copy of the tool, but at the very least I needed to be able to print a copy for all team members (and the client). This meant that I would be able to produce both a hard copy or an electronic copy, e.g., an Acrobat .pdf file.

With cost being a major consideration, I originally began using a free tool called "Playground" from Object International (www.OI.com) to perform the object analysis and design. This worked well as the learning tool it was intended to be. By remaining relatively simple, it allowed me to focus on designing the objects and not on using a new software product. However, I rapidly encountered its limitations both in its ability to handle large models and its ability to support the necessary documentation requirements. This tool did allow copying the model to the clipboard and pasting the diagram into MS Word. It did not support exporting to another file format nor did it allow documentation of any kind other than as graphic blocks.

After approximately three months of use with Playground, I was forced to buy two copies of Rational Rose. The original Playground model had to be manually entered into Rational Rose. Rose was selected primarily because the other object modeler had a script that would run within Rose and generate ODQL from a Rational Rose data model. I found Rose's price to be comparable to most other modelers such as TogetherJ, about $2,000 for a single user version.

Coordinating development of the object model between two remote modelers posed some problems. Reconciling versions of the data model became time consuming. Rational Rose provides a visual-differencing tool; however, we still had to manually run through the entire model checking each component identified as being changed. We subsequently learned to constrain each of our development efforts to separate sections (packages) of the model. A versioning system for Rose would have been appreciated, especially one that worked over the Internet.

Our final model resulted in over 195 classes. Rose readily handled this size producing a model file of about 3 MB, including all documentation and also the use case models.

Coordination and Documentation

Coordinating the model development among the modelers and the programmers relied principally on generating hardcopy (or Acrobat pdf) output for use and reference by the programmers. They did not have a copy of Rational Rose. Printing hard-copy documentation consisted of using the Rose capability to print narrative documentation about the model. This works reasonably well except that each time documentation is generated it starts the document from scratch. This means that any edits to the documentation must be done within Rational Rose itself. Despite the ability to print the narrative, a large part of the model is the diagrams showing the relationships of the various classes. Rose does not have a built in capability to create a document with all the diagrams copied into it. As a result, creating documentation consisted of manually copying each diagram and pasting each one into MS Word for printing. Knowledge of Visual Basic script would allow the creation of a script that would automate this process.

A new enhancement to the latest version of Rational Rose will definitely improve coordination among team developers. This enhancement enables the automatic creation of a web version of the model created in Rose. This is a very sophisticated output that mimics the interface of Rational Rose itself, allowing the viewer to explore the model in a tree view interface and being able to "drill down" to individual class and property documentation. An excerpt of our model in web output format is at http://www.vwi.com/public/Demo/it/v53aMPDMS/v53ahfpa.htm.

While Rational Rose is not the only modeler on the market, the addition of the Integration Kit on the Jasmine ii Beta CD makes Rose the only choice for modeling Jasmine objects, in my opinion. The integration kit provides the ability to generate ODQL directly from the Rose model, and also the ability to import a Jasmine model out of the database into Rose. Using this kit required a slight modification to our existing model in terms of method of documentation, however the entire model of over 195 classes was modified within a single man-day.

Schedule

The requested project deadline was about 199 days, however this was changed in the initial contract to be 261 calendar days. The original schedule was divided into four major phases:

1. Requirements development and analysis
2. Defining the Architecture and Object Design
3. Stages 1 through 3 of coding
    - Database entry screens
    - Output and Analysis
    - Loading the data into the application
4. Product release.

This schedule was subsequently changed to reflect a series of iterations. This was primarily due to the object design phase falling behind. Through the use of iterations, with each reflecting a complete but separate aspect of the application, OOD could continue on one aspect while coding began on those areas where the OOD was felt to be complete. The revised schedule reflected 349 calendar days with the following major phases:

1. Requirements Development
2. Iterations #1-#5 covering different subject areas (building, organizations, analysis and charting, etc.)
3. Loading the data and installation on the client machines.

The following shows the latest time estimates for the major tasks. The table shows how a substantial amount of time was devoted to the modeling effort itself. It must be noted that the efficiency of object modeling allowed for what I would consider a minimal amount of time devoted toward the actual coding of the solution. The above 700 man-hours for ODQL (Object Data Query Language) equates to about 3.5 hours per class. This is very effective considering the level of complication each of these classes represent. It must also be noted that while our programmers were relatively new to ODQL, due to its similarity to C and its adherence to object metaphor the programmers became proficient very quickly.

Task	**Latest**
OOA/OOD	13 months
Screen Design	2 months
Screen Coding	520 hours
ODQL	700 hours
Data Entry – Deployment	TBD

Determining the initial business requirements and relationships usually begins with a meeting or brainstorm session with the client. As part of the initial analysis phase, I had scheduled two separate brainstorm sessions. Each session was to last two days each. Attendees included 2 principal users, 3 developers, and 6 additional Subject Matter Experts (SMEs) (4 of which were contractors involved in the master planning process). These sessions were not nearly as productive as I had intended or scheduled. A main reason is that although it is difficult to pull so many people out of their normal business schedules for too long, and it was an added expense to the contract for the contractors in attendance, the sessions were too short. A week should be scheduled for each session. We spent nearly the entire two days working through the normal "storming" phases of large committees. I also had to provide an initial overview of the basic concepts of object modeling and its syntax. By the time we had progressed to becoming productive in identifying the project needs and objects, the session was over.

Estimate

The obvious lesson learned was that cost estimates should not be determined predominantly by senior leadership without detailed review of the requirements. Software estimation tools should definitely be used to assist in preparing estimates. Results from such tools may also help to reign in requirements that are beyond the client's budget.

As a point of note, the Construx estimator (see

Tools section below) indicated a cost of nearly $2.5 million, we started with $125k. My first estimate was $160K, with the final estimate being $350K. It must be noted that the Construx estimator is based upon a final software product, while our requirement was for a prototype. Clearly, our initial estimate was too optimistic, even for a prototype.

If software is to be developed on a fixed price type of contract, then requirements development should be executed separately based upon a time plus cost method. This way both client and developer will have reasonably agreeable perspectives on the end goal.

Test Data

As the project developed to the point where we had coded classes and screens capable of interfacing with the Jasmine database, we needed data to adequately test the operations. Creating new data from within Jasmine Studio can be very convoluted since a typical business object entails layers within layers of object relationships. Working within Studio it can become difficult to figure out where to begin creating an object and rapidly creating several objects requires a lot of manual effort. This is not a reflection upon Studio as a tool. It is very effective for navigating through the numerous layers of object relationships and for creating an object. However our model called for several objects to be in existence prior to other objects being able to be created. A typical business model is very in depth with several layers of complexity.   Supporting convenient and rapid data entry into the business model is the reason for custom designing screens for the client. However, we also needed a simple process for getting data into the system, and we did not yet have screens to do so. We wanted to load several hundred objects from a legacy xBase file and did not want to resort to manually reentering all of this information. To resolve this we turned to the use of load-unload files. Unload files allow for the rapid transfer of data between two Jasmine databases. They are also used to transfer the class models from one site to another. We hoped to use this method to quickly load the Jasmine model with sample test data.   The Jasmine documentation is complete enough to understand how to make the call to the two programs to perform these functions. However, learning the specifics for modifying the unload files to "paste" in new data did require some trial and error.

The process we used to quickly get data into Jasmine is as follows:

For each class that you want to upload object data, create one or two complete objects, from within Jasmine Studio. This is so that when you create the initial unload file you can determine the correct order of the attributes, and also have a starting point to begin loading data. The quality of these initial entries is not important since we erased it all later in the process. A sample of a typical layout is shown in Figure 2. If the flag to include class structure is included in the call to unload, this will be displayed in the unload file sandwiched between the class name and the start of the data columns.

Unload the classes to which you want to upload data. If the objects include references to other objects, then those objects' classes must also be unloaded. If this step is not taken, then you will have to ensure you are using the correct OIDs for each of those references when you create your own objects in the new load file. It is easier to manage the relationships if you are creating the OIDs for all of the effected objects at the same time.

Reviewing the ASCII unload file we determined that for each set of objects listed under each class the OIDs are bracketed with less-than signs on the left and greater than signs on the right, for example: <1025>. These OIDs are not enclosed within quotation marks. The other important note is that exported OIDs always begin with <1025>. This means that new objects being added through this unload process should always start with <1025> and increment upwards from there.

After quick review of the unload file, the location and layout for the class can be determined. We then used MS Excel to create a spreadsheet to duplicate the columnar layout of the class. Excerpts from our spreadsheets are shown in Table 1 and Table 2. The first column is the OID of each object. Column three shows how a reference to another object is displayed. A complete reference consists of the <Class Family::Class Name::OID>. The last two columns shown in the spreadsheet are not transferred out to the unload file, but are used to easily create our object reference strings (refer to Table 2 to see how the formula's utilized these columns). Unload formats for types of attributes are: OIDs must be bracketed in < and > without quotation marks; numeric attributes must not have quotation marks; character strings must be within quotation marks; date types must indicate the attribute type by specifying the word 'date', but without the quotes, followed by the complete date in quotation marks as follows: date"YEAR-MONTH-DAY-AD" to include the hyphens as shown. Year, month and day are in numeric format. Specify "AD". We are uncertain if "BC" would be an acceptable date type.

Once the spreadsheet has been created so as to mimic the correct columnar layout of the unload file, we exported the columns to a comma delimited file (CSV) format. We then opened the CSV file and pasted the results into the previously generated Jasmine unload file. Shown in Figure 2 is a sample of the output contained in a typical unload file. The first line identifies the format of the ULD file. The second line identifies the class family ("VWIrose") and the class name that is to follow with its data. The critical item to pay attention to for the purpose of pasting in your new data is the last number on this line: 120, shown boxed and highlighted. This number indicates the number of data objects that will follow in the ULD file for this particular class. Thus in this case the 120 indicates that 120 data objects are being added to the Jasmine database.

After the above step we then deleted the classes from the Jasmine database. This accomplishes two important items: it removes any objects from the database precluding duplicate objects from being created, and two it ensures that our OID number sequence starting with 1025 does not cause problems. If a load is attempted without removing any previous objects, the references you created by referring to a particular OID number may not be correct, since Jasmine will dynamically assign new OIDs if the requested OID already exists. A Jasmine command to "ZAP" the database would be appreciated in support of this process. Anyone familiar with dBase or its xBase relatives knows that the ZAP command deletes all data from a database while retaining the structure (in this case the class) intact.

After the applicable classes have been deleted, a Load command is performed including the -s flag to pull in the class structure. This results in the class automatically being defined and built prior to importing the new data.

Table 1: Excel spreadsheet showing creation of data for pasting into unload file

oid	KeyID	MEPRSRefCode	DMISID	RefOID	DMIS OID
<1025>	1	<VWIrose::meprsRefCode::1025>	<VWIrose::dmisid::1733>	1025	1733
<1026>	2	<VWIrose::meprsRefCode::1028>	<VWIrose::dmisid::1733>	1028	1733

Table 2: Same excel file as above but showing the formulas for building the data

oid	KeyID	MEPRSRefCode	DMISID	RefOID	DMIS OID
"<"&TEXT(B101+1024,0)&">"	1	"<VWIrose::meprsRefCode::"&G101&">"	"<VWIrose::dmisid::"&H101&">"	1025	1733
"<"&TEXT(B102+1024,0)&">"	2	"<VWIrose::meprsRefCode::"&G102&">"	"<VWIrose::dmisid::"&H102&">"	1028	1733

```
Jasmine/unload_format_1.0.0

"VWIrose","startFYTimePoint",4,1,0,0,0,0,120,"====== CLASS startFYTimePoint ==== from TimeReference Package ====="

"VWIrose","timerClass"

"oid","DateTP","TimeTP"

<1025>,date"1900-10-1-AD","00:00:00"

<1026>,date"1901-10-1-AD","00:00:00"
```

Figure 2: Sample of unload output for textual data

Tools

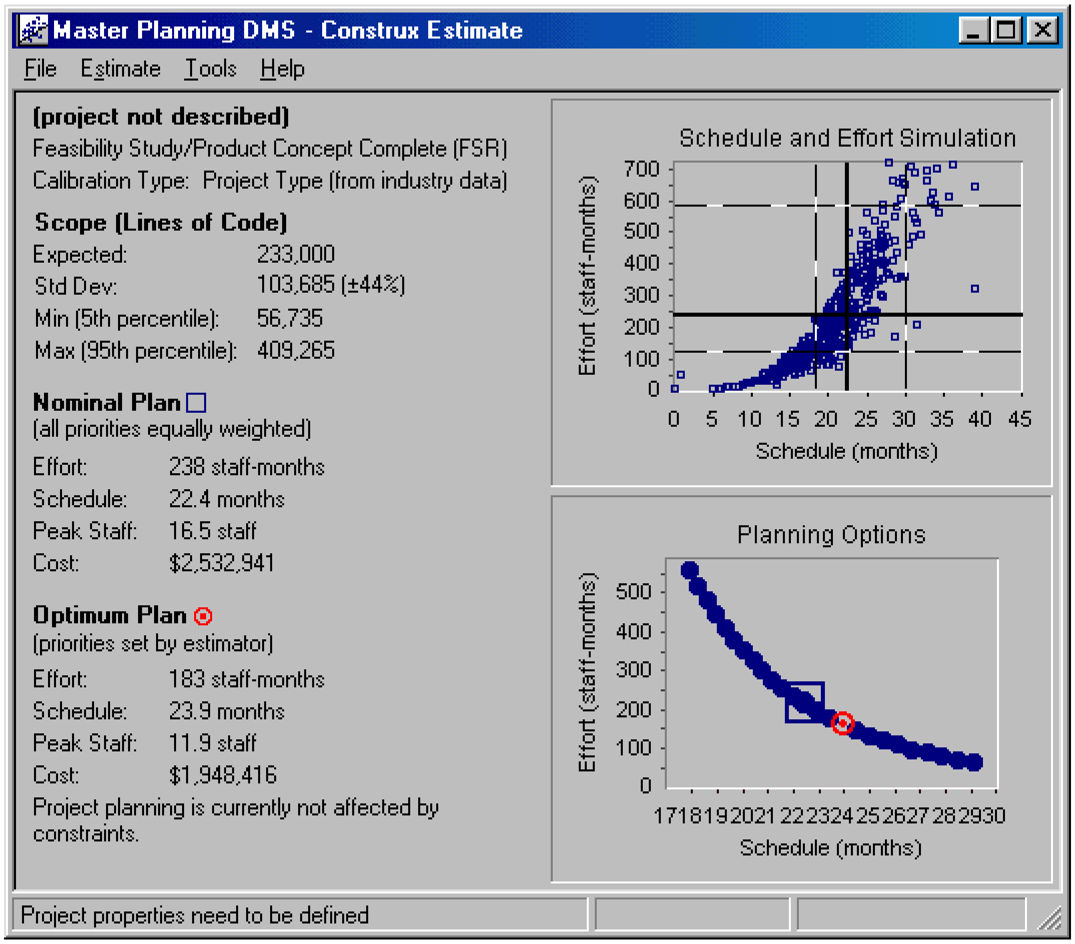
Over the course of the project several software tools were used or desperately wished for. In some cases we created our own tools to meet the needs. Such needs included: cost estimating, ODQL generation from our data models, incorporating the programmers' ODQL back into the data model (a comparison tool), and replication of ODQL source (never did find a solution for that one). Some of the tools we used are described below:

Jasmine Workbench, www.InfoPike.com. This tool provided a GUI utility to perform basic but essential tasks in Jasmine, such as: creating new or deleting class families, creating and extending stores, CODQLIE the ODQL editor in a GUI window, plus others. This tool greatly sped the completion of many necessary tasks. InfoPike is working on a new Jasmine ii version of this tool.

ODQL generation. Generating ODQL directly from the data model diagram avoids an opportunity to make mistakes. Our development partner Archaeotech had developed a script to accomplish this one way action. This Rational Rose script would read a UML model and generate correct ODQL code for compiling into Jasmine. The latest Jasmine ii Beta CD contains the Integration Kit for Rational Rose which allows generating ODQL from a Rose model. The ODQL generated will run in v1.2 Jasmine with a specific exception. The kit does not truncate long descriptions stored within a Rational Rose model, (CA is aware of this and has initiated a resolution). As a result, the code will crash whenever a description line longer than approximately 200 characters is encountered. I developed a small utility in CA-Visual Objects, available at www.Hungry-Hippo.com (free), that will correct this problem by creating a copy of the ODQL script but with truncated descriptions.

Jasmine Model comparison. Although we could generate ODQL code successfully from our UML model, we needed a way to check it against the ODQL code that was actually being written by the coders. Again I developed a utility in CA-Visual Objects (free at www.Hungry-Hippo.com) that we used to compare two Jasmine class families. To support this comparison we created two class families: one from the Rose model and the one being coded. This utility can then print two exception reports, one showing missing classes, and the other showing differences in properties. This allowed us to update the Rose model to reflect new methods and attributes being added by the programmers as part of the implementation process.

Cost Estimating: Construx™ Estimation Software (free) www.Construx.com. This tool enables quickly estimating the cost of a software project based upon a database of other projects of similar size, complexity and demands for timeliness. You begin by entering estimates for the size and complexity of your project based upon lines of code expected, or number of classes, plus a timeframe for project completion. The software then shows similar projects displayed on a curve of time versus cost (see Figure 3), with your own project's estimate indicated on the graph. While we did not actually use this software for preparing our contract bid, it did help to assuage our egos that we were working under limitations. Note our original scope was $125,000 compared to the Construx estimate of over $2.5M.


Master Planning DMS - Construx Estimate

File Estimate Tools Help

(project not described)
Feasibility Study/Product Concept Complete (FSR)
Calibration Type: Project Type (from industry data)

Scope (Lines of Code)

Expected:	233,000
Std Dev:	103,685 (±44%)
Min (5th percentile):	56,735
Max (95th percentile):	409,265

Nominal Plan □
(all priorities equally weighted)

Effort:	238 staff-months
Schedule:	22.4 months
Peak Staff:	16.5 staff
Cost:	$2,532,941

Optimum Plan ◎
(priorities set by estimator)

Effort:	183 staff-months
Schedule:	23.9 months
Peak Staff:	11.9 staff
Cost:	$1,948,416

Project planning is currently not affected by constraints.

Project properties need to be defined

Figure 3: Construx estimate for project.

Summary

This paper provided a broad review of my experience with my first Jasmine project. Specific lessons learned from this experience were highlighted as points of interest. Specific tools used in the project were identified as well as an extensive reading list for new object database project managers. During this project we found that Jasmine provided us a fully capable tool for creating a modern application that would meet the client's demands. Due to its full support for multimedia of all types, and its fully computational language support we did not have to spend time trying to solve basic storage problems. Our full effort was focused upon the solution of the business problem and modeling the business relationships, and not on working around the technical shortcomings of a particular storage platform. Providing the capability to store and manipulate photographs, drawings, CAD documents and full numeric and text data was as simple as embedding preexisting Jasmine classes within our new custom business classes. For future complex solutions incorporating the modern requirements of storing video, photos, and other graphics while permitting the modeling of flexible and adaptive business operations, I would readily recommend Jasmine as the base platform for development.

Biography

Bill Cross is a Senior Functional Analyst and Health Facility Planner for VW International, Inc., an engineering and management firm. As part of a 17 year career in medical facility planning and construction he acquired skills in dBase, Clipper (8 years), dbFast and Visual Objects (5 years). He has been following Jasmine since its prerelease stages. At the time of this writing, he is currently working on adapting a legacy Clipper application to new functional requirements for a Texas utility company.

Reading List

Blaha, Michael. *Object-Oriented Modeling and Design for Database Applications.* Upper Saddle River, NJ.: Prentice Hall, 1998. ISBN 0-13-123829-9

Coad, Peter. *Java Modeling in Color with UML.* Upper Saddle River, NJ.: Prentice Hall PTR, 1999. ISBN 0-13-011510-X

Coad, Peter. *Object Models Strategies, Patterns, & Applications.* Upper Saddle River, NJ.: Yourdon Press, 1997. ISBN 0-13-840117-9

Fowler, Martin. *Analysis Patterns Reusable Object Models.* New York: Wiley, 1997. ISBN 0-201-89542-0

Fowler, Martin. *UML Distilled.* New York: Addison Wiley, 1997. ISBN 0-201-32563-2

Khoshafian, Setrag. *The Jasmine Object Database.* San Francisco, CA.: Morgan Kaufmann Publishers, Inc., 1999. ISBN 1-55860-494-4

Kruchten, Philippe. *The Rational Unified Process, An Introduction.* New York: Addison-Wiley, 1999. ISBN 0-201-60459-0

McCarthy, Jim. *Dynamics of Software Development.* Redmond, WA.: Microsoft Press, 1995. ISBN 1-55615-823-8

McConnell, Steve. *Software Project Survival Guide.* Redmond, WA.: Microsoft Press, 1998. ISBN 1-57231-621-7

Quatrani, Terry. *Visual Modeling with Rational Rose and UML.* New York: Addison Wiley, 1998. ISBN 0-201-31016-3

Reingruber, Michael. *The Data Modeling Handbook.* New York: Wiley, 1994. ISBN 0-471-05290-6

Silverston, Len. *The Data Model Resource Book.* New York: Wiley, 1997. ISBN 0-471-15364-8

Straley, Stephen. *The Encyclopedia for Jasmine vol. 1.* San Francisco, CA.: Sirius Press, 1998. ISBN1-890726-09-5

Taylor, David. *Business Engineering with Object Technology.* New York: Wiley, 1995. ISBN 0-471-04521-7

Taylor, David. *Object Technology, A Manager's Guide.* New York: Wiley, 1998. ISBN 0-201-30994-7