Leveraging Object-Oriented Development at Ames

Greg Wenneson and John Connell
Software Engineering Process Group
Sterling Software at NASA Ames

ABSTRACT

This paper presents lessons learned by the Software Engineering Process Group (SEPG) from results of supporting two projects at NASA Ames using an Object Oriented Rapid Prototyping (OORP) approach supported by a full featured visual development environment. Supplemental Lessons Learned from a large project in progress and a requirements definition are also incorporated. The paper demonstrates how productivity gains can be made by leveraging the developer with a rich development environment, correct and early requirements definition using rapid prototyping, and earlier and better effort estimation and software sizing through object-oriented methods and metrics. Although the individual elements of OO methods, RP approach and OO metrics had been used on other separate projects, the reported projects were the first integrated usage supported by a rich development environment. Overall, the approach used was twice as productive (measured by hours per OO Unit) as a C++ development.

Combining Object Oriented (OO) methods with a Rapid Prototyping (RP) approach supported by a rich development environment holds promise for highly productive development done right the first time. This combined Object Oriented Rapid Prototyping (OORP) approach was used on several projects at NASA Ames and measured over twice as productive as C++ productivity metrics collected by Capers Jones of Software Productivity Research. These projects were supported by training, consulting and mentoring from the Software Engineering Process Support Group (SEPG). Conclusions and lessons learned are presented here for two of the projects now in production: NASA Science Internet (NSI) Service Request (NSR) Tracking System and SoftLib, a reusable software library management system, internally developed by the SEPG.

SEPG Presence, Supported Methods and Approach

The Sterling SEPG acts as a software and process clearinghouse while providing no or low cost engineering and software process, methods and consulting support for contract staff and NASA scientists at Ames Research Center. The SEPG locates, adapts and champions new technology and productivity improving methods primarily as a demand driven resource. We promote several primary methods, approaches and tools which we support by providing training, process guidebooks, consulting, tools and procurement assistance, analysis and design assistance and project mentoring. When requested, we will approve methods and approaches if they are defined by published works and leveraged by available tools, but we prefer a more proactive support presence. Our preferred methods and approaches are:

- The integration of Coad-Yourdon object-oriented analysis (OOA) and design (OOD) methods with a rapid prototyping development approach
- OO Software sizing metrics
- High-level visual-programming development environments.

The Coad/Yourdon (C-Y) methods were selected because they are moderately simple, easily taught and provide consistent analysis and design representation. During this last year with

newly published works by Booch, Yourdon and others, object methods are converging and borrowing the best from each other. Thus Ivar Jacobson's Use Cases and other current approaches are being incorporated into the methods we support. We find these methods and approaches are scaleable for small and large project size in simple to complex problem domains.

The SEPG supported approach is to use C-Y OOA/OOD methods [1, 2] combined with the formal *Object Oriented Rapid Prototyping* approach defined in the new Yourdon Press book by that title [3]. This involves evolutionary development with refinements based on feed back from customer hands-on experimentation during approximately one to two week iteration cycles. The process model for this approach is shown in Figure 1. The identification of customers (requirements owners), their level of involvement and their buy-in are obtained up-front. An initial analysis and an OO model are produced in the first few days of the project for early project planning and then iterated concurrently with the prototype through many incremental additions and refinements. Formal inspections of requirements and design specifications occur at two or three points during this evolution:

- Before prototype development
- After user approval of prototype, before tuning
- Any other time the development team feels a need to resolve emerging design issues.

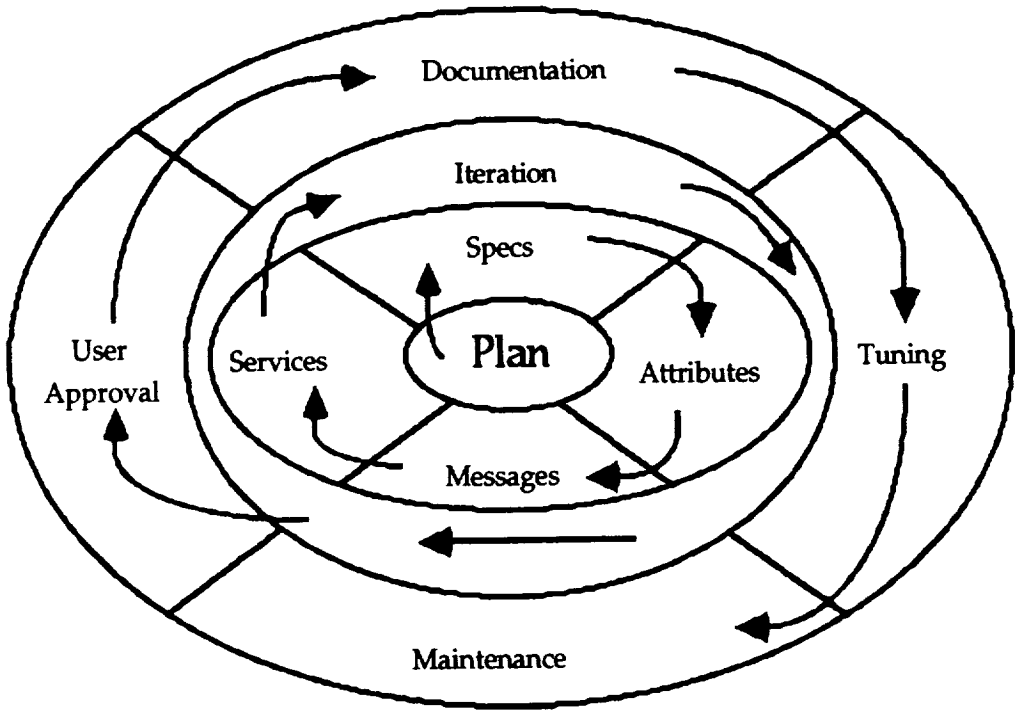


Figure 1 Object-Oriented Rapid Prototyping (OORP) Process

We have been experimenting with some new OO sizing and estimating metrics at Ames. These metrics are similar to those presented by Lorenz [4] but were actually derived as a modification of Dreger's Function Point Analysis [5] adapted to OO methods. The OO Unit metrics were first published in a paper by Connell and Eller in 1992 [6]. OO Unit metrics for components (classes/objects) and services (methods) are given OOU counts depending on the number of attributes in the object. An object with 8 attributes is of average complexity and has an OOU metric of 5. Each of the services would also count at 5 OOUs. Services have different counts

for add/modify/delete, output, computationally intense, and system service but are clumped here for simplicity. External Entities are the sources and sinks of a Source Sink Diagram which defines the system boundaries. External entities receive counts dependent on the number of interfaced objects in the system. Figure 2 provides guidelines for determining OO Unit metrics counts.

OO Units provide an advantage over the Lorenz sizing metrics in that they allow for differentiating object classes according to size, depending on the number of attributes, services, complexity of services, and external interfaces. The differentiation scale is based on a similar scale provided by Dreger and Capers Jones

	Simple < 7 Info Items	Average 7-14 Info Items	Complex > 14 Info Items
Component	3 OOUs	5 OOUs	8 OOUs
Service	4 OOUs	5 OOUs	6 OOUs
External Entity	< 3 Components 7 OOUs	3-5 Components 10 OOUs	> 5 Components 15 OOUs

Figure 2 Object-Oriented Unit Metrics Matrix

Using the C-Y OOA/D methods and Connell/Shafer rapid prototyping approaches, an early estimate of total effort can be made from the initial analysis and OO model generated at project startup. The OO unit metrics are counted from the initial model using the number and complexity of the objects, services and external interfaces. The final delivered application usually grows during prototype iteration to three times the size of the initial model. The estimated times to develop the initial prototype and then the fully deployed application are dependent upon the implementation language and environment. In our estimates, we used a figure of 4 hours to implement an OO Unit, equivalent to the figure Dreger uses for Objective C and Smalltalk. We reasoned that a powerful visual programming environment would be at least as effective as ObjectiveC and Smalltalk. Our project's end results produced figures equal to or better than that, 4 hours per OOU for one project and about 3 hours to deliver an OOU on the other.

We recommend use of high level visual programming environments for development and iteration of rapid prototypes. Ideally, a powerful development environment would provide integrated capability to manipulate GUI, control, functionality and data management abstractions at a higher level than coding in a 3GL. This is still the holy grail of development environments. While waiting for that future momentous unveiling and heeding the current (Summer 1993) call of requirements, we evaluated numerous vendors and selected Sybase's GainMomentum product as a development environment which met selection requirements. GainMomentum (here after referred to as Gain) provides object-oriented visual development tools for RDBMS access, graphic user interface, and user defined objects. Gain also has extensive function libraries, a good debugging capability, and a 4GL scripting language (GEL - Gain Extension Language) to augment visual development tools. Programming in C/C++ code is generally not required. There is an instant context switch from edit to run mode and standalone run-time executables can compiled when an application is complete to restrict user access. Gain was available for Unix environments only, though recently a Windows version was released.

For analysis and design modeling support we used the drawing and data dictionary capabilities of Iconix's ObjectModeler for the Macintosh. Although ObjectModeler can generate code templates, we only used the drawing capabilities of the tool. Consequently, the object models and the tool's capabilities were not integrated into the developers' environment. One project elected to use a Macintosh drawing tool with just as effective results.

For both projects, SEPG members acted as external consultants, trainers and mentors. Just in time training was provided in C-Y OO methods, Rapid Prototyping, development tools and management approaches. The SEPG also provided development tools during the early stages of development so that development could get started on the right foot while project startup procurements proceeded concurrently. During early stages of development, SEPG members provided hands-on assistance with OOA and OOD modeling, prototype development, prototype iteration and refinement methods, estimating, and planning support in conjunction with project staff. When staff were completely comfortable with the methods, they assumed all development activities from the SEPG.

The Projects' Specifics

Both the NSI and SoftLib projects were small and low risk. NSI planned for staff at 3 Full Time Equivalents (FTEs) and the SoftLib project was planned at about 1 FTE. Due to personnel and organizational changes, neither project reached their full planned staffing. The NSI project was estimated to take 6 calendar months and the prototype was approved and completed in 7 months. When the approved prototype was delivered, the users required further work which was completed 4 months later. The SoftLib development was initially scheduled to take 11 months and completed on schedule. With the organizational changes, we consider both projects to have completed within projected time and costs. Project effort and metrics are discussed in the next section.

Both of the projects used inexperienced staff assisted by SEPG consultants. The projects were the staff's first introduction to object methods, rapid prototyping, full life cycle implementation, advanced development environments, RDBMS and SQL.

The NSI project developed a new application to manage Internet connectivity requests stemming from world-wide NASA science projects. The development involved creation of two complex data entry forms: the NSI Service Request (NSR) and the Request for Service (RFS). The combination of these two vehicles and supporting data structures provides on-line entry of customer profile and organization data, funding authorization, and Internet service connectivity requirements. The application replaced and integrated manual and ad hoc systems for several groups, adds new functionality and provides the opportunity for further automation.

The SoftLib project re-engineered to modernize an existing reuse library management system. The old system provided a character based front end to a database of metadata about software components. Users had to grapple with the character mode interface to find reusable software descriptions and then locate the actual software outside the domain of the library management system. The new application provides a graphic X-Windows user interface to increased capabilities. Combo-box list widgets now provide selectable keywords and other search parameters. When users find interesting component descriptions in the hit list, the location is presented and they can download the file using another window. The application also provides interfaces to other applications such as a New Technology Database and other reusable libraries including a NASA-wide BBS.

In addition to the commonalty in development approach, inexperienced staff, estimating metrics and visual development tools used, these two projects had certain other elements in common. Both were in environments where users and developers did their work on networked combinations of Macintosh and Sun workstations. The networks extended over many Macintosh zones and Internet domains within the Ames domains. Both applications required intensive user interaction and an interface to an existing relational database.

There were also several differences between the projects in that SoftLib was developed using a very new alpha/beta release of a truly object-oriented version of the Gain development environment, while the NSI project used the current production release. Mentoring on SoftLib was fairly smooth because the project was internal to the SEPG. The NSI project used multiple and conflicting sources for consulting causing some confusion and lost time due to thrashing back and forth between divergent approaches — information engineering versus object-oriented rapid prototyping.

Perhaps one of the primary differences was in user's profiles and expectations. SoftLib replaced a single text based system. The users, although from different application domains, were familiar with a single interface. Whereas on NSI the users were from different functional groups. There was no single application to replace; indeed, many users had evolved their own applications using spreadsheets to support their work. Some of the replaced functionality was being performed by data entry staff. The NSI authors felt that many of the users did not think that they would be using the system.

Access and data security were issues for both projects. NSI's solution was at the network administrator layer—disallow access outside project domains. SoftLib specifically needed to allow access and file download capability throughout the Ames domains but not to outsiders. The initial design was for security daemons, user accounts and client-server pairs for file transfer. The access and security features were written in C due to apparent limitations of the Gain environment. Very late in the development, the entire SoftLib security/file transfer implementation was replaced with an Xmosaic shell with "allow" access capability for the Ames domain and a separate Xmosaic window for file transfer. Distribution and installation packaging were also replaced due to portability problems of executable code to heterogeneous workstations. As a result, no software is required to be distributed to potential users and the developer has greater control over enhancements and problem fixes. All SoftLib capabilities are available (in the Ames domain) through the World Wide Web.

Results and Lessons Learned

The NSI NSR/RFS application is in production and being used. When the approved prototype was delivered, the users were not happy and required 4 additional months of part time development. Most of the users are actually on Macintoshes using MacX for X-Windows emulation, although the system was mostly developed and demonstrated on a Sun workstation. The result is that the delivered system is very different from what the users expected. The system feels slow for this application on this network. Part of the problem appears due to heavy server loading and the earlier version, reduced-capability of Gain data managers. Macintosh client performance is less than half Sun workstation performance due to remapping for MacX screen display. Also screen size and pixel density are very different, giving a degraded look and feel on the Macintosh. NSI Macs are currently being upgraded with larger screens and graphics accelerators. Because an earlier version of Gain was used on this project, much additional GEL scripting was needed for database transaction management.

The SoftLib application has recently gone into production (October 1994) after successful beta testing in August and September. The performance is faster than NSI's application and quite acceptable. It was developed in the newer Gain version and deployed on a different host. As with the NSI project, the SoftLib prototype was primarily developed and demonstrated on a Sun workstation while many users are on Macintoshes. However, with the SoftLib application, the Librarian is promoting the reuse library and is using the colorful, more capable interface as advertising leverage to attract users.

These projects are characterized as successful because they went into production and are being used. They completed within 20% of originally planned schedule and resources. The C-Y OOA/D methods were introduced, learned and used in development. The initial C-Y object class models and OO Unit metrics provided an acceptable basis for project estimating and planning. Data points were generated to calibrate the metrics methods. Connell/Shafer rapid prototyping approaches were used to iteratively generate a hands-on requirements model the users requested and then a deliverable product. A new object oriented development environment was used to produce applications which are fairly easy to change. Preliminary measurement of development time is about 3 hours per OO Unit for SoftLib and about 4 hrs/OOU for NSI. The NSI figures are higher due to the larger amount of GEL and SQL written. These results are from inexperienced developers leveraged by visual development environments. And we had fun!!!

We feel that an OO Unit is very similar to a Function Point as described by Dreger [5]. Dreger (based on work by Capers Jones) provides a list of relative effectiveness of implementation languages including 4GLs. However, Dreger only provides one single-figure productivity metric—an average of 20 hours of COBOL development to produce one Function Point. (Capers Jones [7] declines to give language-dependent single figure metrics. Jones prefers to give high-low ranges for productivity, probably to prevent comparisons in papers like this.) We generated single-figure productivity figures by taking the median of the productivity ranges provided in Dreger's and Jones' figures. Since Jones' and Dreger's figures are given in Function Points per staff month, we assumed 21 working days per month and 6 working hours per day to normalize to hours per FP. From this, we show productivity figures for C (24 hours/FP), FORTRAN (20 hours/FP), C++ (15 hours/FP), Ada (14 hours/FP) and ObjectiveC/Smalltalk (4 hours/FP).

We are not entirely comfortable with our single-figure interpretations of Jones figures. Jones' collected metrics are from a wide range of project types and environments including MIS, military, and system software among others. We feel that today's versions of the languages would permit at least twice the productivity of our medians of Jones ranges. Using that adjustment, C would be 12 hours/FP, C++ 8 hours/FP and Ada 7 hours/FP. These adjusted figures are consistent with the high end of the productivity range Jones does provide for each of the languages. Following that, what our projects with inexperienced developers accomplished in 3 and 4 hours still compares favorably to what Dreger/Jones data shows as 8 hours per Function Point in a standard OO programming language such as C++ or 12 hours in a lower level language such as C.

We feel our productivity could have been even better. We think about 10% of total effort on the NSI project was spoilage due to conflicting advice provided by competing consultants from different organizations. On both projects, productivity was lessened by the steep learning curve of multiple elements (OOA/OOD, Gain, rapid prototyping, SQL and basic development experience). We estimate that overall on-the-job learning constituted at least 30% of total implementation cost on these two projects. On both projects the majority of implementation problems and effort expended were related to overcoming the data managers and database

interface. On NSI, much GEL scripting was written to overcome the earlier version of data managers. On SoftLib, the newer production release data managers are quite powerful, but developers had to struggle with alpha versions and multiple Beta releases. All in all, we estimate it took at least 20% additional development time for each project to overcome the maturing data base interface.

Prototype size growth from initial OO model to delivered system was flat for SoftLib and about 2.5 for the NSI system. Both systems were estimated to grow to three times the OOU counts from the initial OO model to the final system. The initial SoftLib model had an unrealistically high OOU count because the graphic widgets were modeled as separate objects with services rather than services of objects. A remodeling of the SoftLib initial OO model produced a 25% lower OOU count with an actual growth of 0.5 to delivered system. The SoftLib growth was incorrectly estimated because the SoftLib model was a detailed and almost complete model of an existing application rather than an initial OO model of a future system. The actual hours needed to complete SoftLib were also less than half that initially estimated, partially due to learning from the NSI experiences.

Reuse was minimal due to the mismatched capabilities of the development environment versions and the different application domains. The overall application framework and a few of the GUI widgets were reused between the two projects. With a bit more care, several of the NSI object classes (person, organization, etc.) might have been reused within SoftLib. Many of the NSI's classes hold the possibility for future reuse in any resource management system.

In SoftLib, the Gain development environment allowed easy modification of the applications. Because very little code is written outside the development environment, the production version is still as flexible as the prototype was during iteration. The small amount of C code written to provide SoftLib security and controlled file transfer was easily replaced using the more portable Xmosaic's file transfer and security features.

There was some learning transferred from one project to the other. The SoftLib developers were able to make some use of NSI lessons learned. The different versions of Gain data managers prevented more knowledge from being transportable. The SEPG members consulting on the NSI project also consulted on the SoftLib project. That connection was lost as the project team members assumed all responsibilities from the SEPG.

There were also some harder to measure productivity loss factors. The inexperienced developers made some mistakes that more senior software engineers might have avoided. One side effect of inexperienced prototypers was the hesitation to demonstrate a prototype that didn't appear excellent. This resulted in fewer iterations and less frequent user feedback. User commitment to requirements approval was difficult to obtain. On SoftLib there was one primary developer. With better initial team building and work partitioning, communication would have improved and the primary developer's workload lessened. Both projects had to pick between a less capable GainMomentum version 2 or an in-development alpha/beta version 3. There were many handicaps to overcome with either choice. Additionally, the inexperienced developers were not always amenable to the mentoring available from the SEPG. This is because the application was their first masterpiece and suggestions and proposed alternatives were often perceived as criticism and therefore not well received.

A major lesson learned from these two projects relative to the application of the Connell/Shafer rapid prototyping approach is that delivering a system (Macintosh) with a different look and feel from the user approved system (Sun) diminishes much of the requirements stability gained from

prototype iterations. In order to achieve requirements completeness, correctness, and exactness through rapid prototyping, the following must occur:

- real requirements must exist
- correct identification of user representatives in a development plan
- establishment of requirements ownership in a development plan
- user commitment to prototype review and approval as planned.

Execution of these basic rapid prototyping principles was flawed on both projects, resulting in some user dissatisfaction. Experienced rapid prototypers know that successful rapid prototyping is an evolving team-based process owned mutually by users and developers. The Space Station Centrifuge project, for instance, proved that the OORP approach can be used to overcome group dynamics or political fragmentation problems if users become sufficiently involved in prototype iterations. On the Centrifuge project, solid requirements definition was achieved in 14 iterations over a 10 week period with approval from 100 users. These users were in three different groups (operations, controls, and human factors) each competing for system resources and requirements implementation.

Good News

These projects were sized and scheduled using estimates derived from OO metrics applied for the first time to real projects at Ames. A conservative factor of four hours per OO unit was used for NSI project estimating. The actual productivity figure is just about that. On SoftLib, we used 2 hours per OO unit based on an preliminary estimate of the NSI metrics and hopes for the more mature version of Gain. Preliminary figures indicate a productivity figure at about 3 hours per OO unit. With the results and the offsetting productivity losses mentioned above, we feel the metrics have been initially validated and will continue to be used and refined. From previous and concurrent experience with other prototyping tools, we feel the metrics can be generalized for the entire class of visual-programming in very high level rapid prototyping tools similar to GainMomentum. These kinds of tools are much faster than procedural languages such as C and FORTRAN. They measure several times faster than OO languages such as C++, and hold promise to be significantly faster than OO development environments such as ObjectiveC and Smalltalk. On our projects, when such tools are combined with a formalized approach to OORP, the development time (with inexperienced developers on first time projects) has been measured at equal to any other approach we customarily use. If we adjust our project's productivity's by the estimated losses for learning curve and tool problems, we have an approach about twice as fast as the figures put forth by Dreger for ObjectiveC and Smalltalk.

The OO paradigm is difficult for many developers to master. We have found at Ames that non-complicated modeling methods assist developers in learning and users in understanding. Our modeling activities provided an easy way to depict the initial requirements and explain them to the user before the first prototype was started. The model also acted as an alternate design mechanism with the alternatives or not-yet-built components shown in a different color or otherwise called out. The modeling activities paralleled or led the early development; however, once the major components and inheritance were established, the modeling activity fell to a lower priority. Subsequent metrics determination required the model to be updated and reviewed—which should have been done concurrent with development.

The services provided by the SEPG proved to be valuable and particularly necessary for new developers. The learning curve was too steep for the inexperienced staff to contemplate without the training and on-project consulting provided by SEPG members. At Ames, the SEPG advises, rather than controls, projects. This means that staff may always feel free to ignore

SEPG advice. It has been found that staff are much more likely to heed the advice when it is perceived as free help rather than criticism.

In the course of these two projects a happy accidental discovery was made: there need be no difference between a good Coad/Yourdon OOD object class model and a good RDBMS schema. A mapping can be done such that each object class on the OOD model maps to a table in the database and all required database tables are modeled as object classes. This mapping is possible because the Coad/Yourdon methods work very well for data oriented applications. The methodology guidelines for identification of good object classes map well to normalized RDBMS tables. This is not to say that these methods do not work well for other kinds of applications. One of the most successful applications of OOA/OOD and OORP at Ames is the development of real-time data acquisition software for the new 250,000 LOC Standardized Unitary (wind tunnel) Data System.

Summary

The Rapid Prototyping approach combined with Object Oriented methods and leveraged by visual programming development environments show solid promise to significantly improve development productivity while generating the system the users request. Although the productivity metrics are preliminary and based on a few data points, it appears possible to easily exceed the productivity compared to creating an application with ObjectiveC or Smalltalk. The projects' productivity measures about twice as effective compared to the high productivity range of Dreger/Jones' C++ metrics. We have also shown it is possible for less seasoned engineers using these approaches and assisted by skilled mentors to exceed the productivity of seasoned developers using less effective techniques. We look foreword to measuring fully experienced developers using these highly leveraged environments.

On future OORP projects, there are some things we will do differently, as a result of these projects. We will strive to make sure that we always deliver the system the users really approved, and not slip in a new, unapproved look and feel for delivery! We will pay more attention to psychological factors in dealing with inexperienced staff and uncommitted users. We also need to keep our OO models in better synchronization with the development activity; perhaps that can identify more intentional reuse opportunities. We will try to be more thorough in assuring that original plans are carried through to ensure that users' needs are truthfully identified and responsibilities met.

We would like to compare our metrics to other OO projects in different domains and environments. We used project data captured by Lorenz and Kidd [4] to do a rough comparison to Smalltalk and C++. Their averaged data indicates Smalltalk productivity of less than 1 hour per OOU and a lower productivity for C++ at 3 hours per OOU. This three-to-one ratio is consistent with the Jones and Dreger data. The Lorenz data are from only a few projects but imply a higher productivity than our projects. However, as with the Jones data, we would need more contextual information about developer experience, environment capability and accuracy of the collected data to gauge the comparison and possibly the leveraging effect of different methods on productivity.

There are some other things these projects have caused us to think about, but we have not as yet come to any conclusions. We need to devise some more efficient means for providing expert design guidance to projects so that guidance is heeded more consistently. We need object oriented design and quality metrics in addition to sizing and estimating metrics. We also need object reusability guidelines and metrics. Furthermore we wonder if it would improve

application of SEPG services and better serve the customer if we withdrew SEPG support to a project rather than compete as one of several consulting sources.

Bibliography

1. Coad, P. & Yourdon, E. *Object-Oriented Analysis*, New York: Yourdon Press (Prentice-Hall), 1990, 1991.
2. Coad, P. & Yourdon, E. *Object-Oriented Design*, New York: Yourdon Press (Prentice-Hall), 1991.
3. Connell, J. & Shafer, L., *Object-Oriented Rapid Prototyping*, New York: Yourdon Press (Prentice-Hall), 1995.
4. Lorenz, M. & Kidd, J., *Object-Oriented Software Metrics,* New York: Prentice-Hall, 1994.
5. J. Brian Dreger, *Function Point Analysis*, New York: Prentice-Hall, 1989.
6. Connell, J. and Eller, N., "Object-Oriented Productivity Metrics", NASA Quality and Productivity Conference, 1992.
7. Jones, Capers, *Applied Software Measurement, Assuring Productivity and Quality*, New York: McGraw-Hill, Inc., 1991.

LEVERAGING OBJECT ORIENTED DEVELOPMENT at NASA AMES

Greg Wenneson and John Connell

SEPG
Sterling Software
at NASA Ames

November 30, 1994

STERLING SOFTWARE

SEPG Experiences, Lessons Learned

Combination of :

- OO Methods
- Rapid Prototyping
- OO Metrics and Estimating
- Leveraged by Tools

SEPG

Supports By:

- **Training, Consulting, Guidebooks and Tools**

"Supported" Methods:

- **Coad-Yourdon OOA/D**
- **Connell-Shafer Rapid Prototyping**
- **OO Metrics and Estimating**
- **HyperCard, JAMM, GainMomentum**

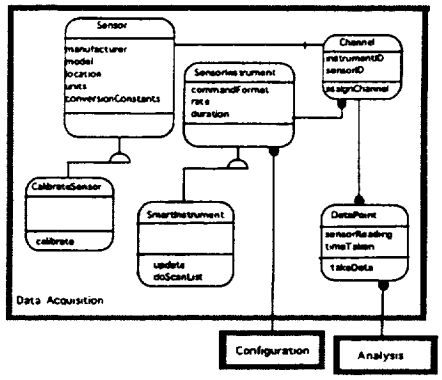
"Approved" Methods ...

Coad-Yourdon OO and...

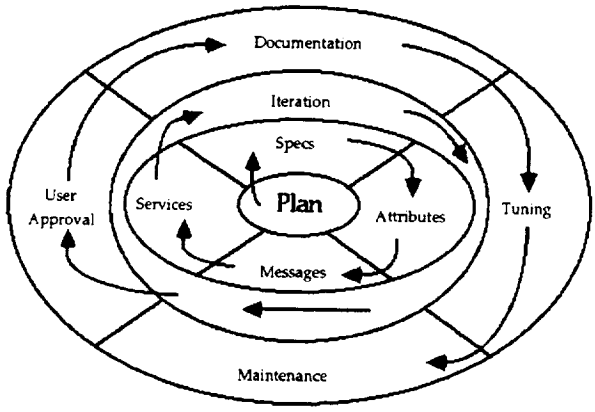
- **Object Classes, Attributes and Services**
- **Subject Layering**
- **Problem, Human I/F, Task and Data Mgt Domains**

plus

- **Source-Sink Diagram**
- **Object Control Matrix**



OORP Process Model



OO Rapid Prototyping

- **Identify Requirements Commissioners**
- **Initial Analysis, OO Model, Estimate and Plan**
- **Prototype Initial OO Model**
- **Iterate with User: ~ 1hr New Functionality**
- **Final Req'ts User Approval**
- **Tune, Re-engineer, Document, Inspect**
- **Acceptance Test and Deliver**

OO Unit Metrics Matrix

	Simple < 7 Info Items	Average 7-14 Info Items	Complex > 14 Info Items
Component	3 CUs	5 CUs	8 CUs
Service	4 CUs	5 CUs	6 CUs
External Entity	< 3 Components 7 CUs	3-5 Components 10 CUs	> 5 Components 15 CUs

- **Object Complexity by Number of Attributes**
- **External I/F Complexity by Number of Objects**
- **Effort = OOUs X Hrs/OOU X 3 (Prototype Growth)**

Visual Development Tools

GainMomentum (Selected in 1993)

- **Object Oriented**
- **GUI Development**
- **Data Management**
- **Function Libraries**
- **4GL-like Scripting Language**

Project Descriptions

NSI Service Request (NSR): Internet Connections

- **Manage, Track and Schedule Resouces**
- **Automate Manual and Separate Systems**
- **Potential 100+ users**

SoftLib Library Management System

- **Reusable Library Component Xwindow Interface**
- **Re-engineer Text Based System**
- **Ames-wide User Base**

Common Factors

- **Both systems Small and Low Technical Risk**
- **Staff Inexperienced; Then Trained**
- **Introduced OO and RP**
- **Introduced New Development Tool GainMomentum v 2. and Beta v3.0**
- **Users Spread Over LANs: Macs and Suns**

Results - NSI

- Planned 6 mo.; Approved Delivered in 7 mo.
- Initial Model ~140 OOUs; Delivered ~550 OOUs
- Estimated 4 hrs/OOU; Delivered ~4hr/OOU
- Performance Not as Expected
- Needed Much Additional SQL
- Delivered Approved Prototype Not Used
- 4 More Months Development
- Problems Not Technical

Results - SoftLib

- Planned 11 mos; Delivered in 11 mos.
- Initial 453 OOUs; Delivered ~ 450 OOUs
- Estimated 2 hrs/OOU; Delivered ~3hrs/OOU
- Some C Code; Replaced by xMosaic
- Newer Version of GainMomentum
- Some Structure and Widget Reuse
- System Performance Satisfactory!

What Worked

- OO & RP Work Well
- OOU Metrics and Estimates Work
- Development Tool Leverages Productivity
- SEPG Assistance Critical to Success
- Productive Development Approach

Improvement Needs

- SEPG Advice Optional
- Steep Learning Curve: 30%
- NSI 10% Multiple Consultant Spoilage
- Following RP Approach
- Non-Technical Issues
  - User Buy-In / Commitment
  - Developer Ego
- Reuse Criteria

Learned

- Deliver what Users Approve ...
- Make sure Users knowledgeably Commit
- RP Can Help Overcome Scattered Users
- Tools Have Warts - Know Them!
- C-Y OOD Obj-Class Model like RDBMS Schema
- C-Y OOA/D Methods Simple and Powerful
- Promoted Methods Leverage Productivity