Structured and Object-Oriented Methodologies: A Comparative Analysis Based on a Case Study

Gary Brian Warren

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
https://louis.uah.edu/honors-capstones/647

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE
Honors Program

HONORS SENIOR PROJECT APPROVAL FORM

(To be submitted by the student to the Honors Program with a copy of the Honors Project suitable for binding. All signatures must be obtained.)

Name of Candidate: Gary Brian Warren

Department: Computer Science

Degree: Bachelor of Science in Computer Science

Full Title of Project: Structured and Object-Oriented Software Methodologies: A Comparative Analysis Based Upon a Case Study

Approved by:

[Signatures and dates]

Honors Program Director for Honors Council

Date
Structured and Object-Oriented Methodologies: A Comparative Analysis Based on a Case Study

by

Gary Warren

CS 495
Dr. Carl Davis
September 2, 1993
ABSTRACT

Since the Garmisch Conference of 1978, software engineering has become the standard way to try to deliver larger software packages both on budget and within time constraints. Software engineering involves two fundamental paradigms: life cycle, or organizational technique, and methodology, or the technique employed to model the real-world system under consideration. The two most critical components of life cycle are analysis and design. Two popular methodologies for employing these techniques are Structured Analysis and Design and the newer Object-Oriented Analysis and Design. Structured Methodology focuses on processing while Object-Orientation focuses on data. In an attempt to determine which was the better methodology, a system was analyzed and designed under both of the methodologies according to specific methods of an author who had switched from the structured to the object-oriented method. Comparative results indicate that both had significant advantages and weaknesses, but that Object-Orientation had intractable flaws mainly as a result of its relative newness and immaturity with respect to definition, meaning, and usage. Though using structured methods is recommended for now, the advantages of object-orientation will make it the better model once it has reached maturity.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>v</td>
</tr>
<tr>
<td>I. Prologue</td>
<td>1</td>
</tr>
<tr>
<td>II. Introduction</td>
<td>4</td>
</tr>
<tr>
<td>III. Methodologies</td>
<td>8</td>
</tr>
<tr>
<td>IV. Results</td>
<td>20</td>
</tr>
<tr>
<td>V. Conclusions</td>
<td>28</td>
</tr>
<tr>
<td>VI. Bibliography</td>
<td>31</td>
</tr>
<tr>
<td>VII. Appendix A</td>
<td>34</td>
</tr>
<tr>
<td>A. Structured Analysis</td>
<td>34</td>
</tr>
<tr>
<td>B. Structured Design</td>
<td>56</td>
</tr>
<tr>
<td>VIII. Appendix B</td>
<td>61</td>
</tr>
<tr>
<td>A. Object-Oriented Analysis</td>
<td>61</td>
</tr>
<tr>
<td>and Design Notations</td>
<td></td>
</tr>
<tr>
<td>B. Object-Oriented Analysis</td>
<td>64</td>
</tr>
<tr>
<td>(PDC)</td>
<td></td>
</tr>
<tr>
<td>C. Object-Oriented Design</td>
<td>81</td>
</tr>
<tr>
<td>(HIC)</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. The Classic Life Cycle 5
Figure 2. The Prototype Life Cycle 5
Figure 3. The Costs of Errors in the Life Cycle 7
Figure 4. A Context Diagram 9
Figure 5. An Entity-Relationship Diagram 10
Figure 6. A Balanced Data Flow Diagram Fragment 11
Figure 7. A Transform Center 13
Figure 8. A Transaction Center 13
Figure 9. A Transaction-Centered System 14
Figure 10. Objective of Object-Oriented Analysis and Design 16
Figure 11. An Object-Oriented Analysis of a Vehicle Registration System 18
Figure 12. The Multi-Component, Multi-Layer Object-Oriented Analysis and Design Model 19
Figure 13. Layer Models of Object-Oriented Analysis/Object-Oriented Design 62
Figure 14. Class-&-Object Specification Template 63
Figure 15. Object State Diagram Notation 63
Figure 16. Service Chart Notations 63
LIST OF SYMBOLS

AD: analysis and design
DD: data dictionary
DFD: data flow diagram
DMC: data management component
ERD: entity-relationship diagram
HIC: human interaction component
OO: object-oriented, object orientation
OOA: object-oriented analysis
OOAD: object-oriented analysis and design
OOD: object-oriented design
OOPL: object-oriented programming language
PDC: problem domain component
SA: structured analysis
SAD: structured analysis and design
SD: structured design
TMC: task management component
Prologue

On the day of November 9, 1979, civilization as we know it came unusually close to its end. That day, the strategic air command had an alert that scrambled our nuclear forces. The reason our forces scrambled was a mistake. It was not a political or military decision though, but a computer mistake. The WWMCCS computer confused a simulated attack as real because of a software fault, and it signaled our forces that the Soviet Union had launched missiles aimed at the United States. The movie War Games, which appeared five years later, though largely over-dramatized, did show with chilling effect that computer errors, even one error on one computer, can be detrimental to us all [17;4].

This incident should be proof enough that computers and their software effect some of the most fundamentally important aspects of our lives. Yet life-threatening situations involving national defense are only a small aspect of our lives into which computers have been introduced. Computers have become invaluable to high-tech medicine, avionics and shipboard navigation, banking, word processing--almost all types of business in fact; we continue to rely on them for technical superiority in the competitive world in which we live. With such a tremendous amount of reliance on computers, it has become increasingly necessary that they operate more reliably both from a safety and economical standpoint.

Two factors are involved in computer performance:
hardware and software. The hardware is the physical, functioning electronic circuitry; it is the computer chips and bus lines. The instruction set that drives the computer is software. By the time it reaches the market, hardware can be considered fairly reliable because of its rigorous development and precise automated manufacturing techniques. Software, on the other hand, is delivered by humans. Therefore it is only as accurate as the skills of the programmer or programmer team that creates it, the methods used to develop it, and the tools available to help the process.

It is not easy to program computers. As John Guttag of MIT explains,

Anyone with substantial programming experience knows that building software always seems harder than it ought to be. It takes longer than expected, the software's functionality and performance are not as wonderful as hoped, and the software is not particularly malleable or easy to maintain... In its purest form, it is the systematic mastery of complexity [9;9].

With such a task before them, it is not surprising that early computer programmers floundered when it came to producing software, especially on the increasingly complex systems being demanded. In fact, by the 1970's, "the computer world had become famous for failures: dangerous system errors, late deliveries, spectacular budget overruns, and abandoned projects" [6;115].
A key turning point in the software development process came in 1978. That year, NATO held the Garmisch Conference, also called the NATO Software Engineering Conference, a meeting of government officials and some of the most prominent people of the international software market. Here, the three major problems of large software development were noted: (1) projects were exceeding their deadlines greatly, (2) costs were over budget, (3) systems were not meeting their expected performance expectations [6;102]. Among its many recommendations on what to do about what by now was called the "software crisis" or "software bottleneck," the conference endorsed the idea of software engineering, coined by a NATO study group the previous year. Software engineering implied that the process of producing software was comparable to other engineering processes [17;5]. The important feature introduced was that programming as purely an art gave way to a methodology with a corresponding discipline [6;127]. So software engineering got its start and has since become critical for producing large software packages.
**Introduction**

A good definition for software engineering today is "the technological and managerial process concerned with the systematic production and maintenance of software products that are developed and modified on time and within cost estimates." Software engineering as a process can best be understood in terms of two paradigms: the life cycle paradigm and the methodology paradigm.

The life cycle paradigm is the larger example into which the methodology paradigm fits. Life cycle really is the organizational technique used to obtain a finished product. It is the way that available methodology and tools are put into use, hopefully to achieve the most efficient construction for a specified product. The classic life cycle is the most common. It is often known as the "waterfall" approach because of the way it looks (Figure 1). In this paradigm, steps are taken in sequential order to get to the final product. Another popular life cycle model is called prototyping (Figure 2). Prototyping is the more rapid method of the two. With it, programmers quickly construct a minimally functional shell of a program that can perform the most required behaviors of the system under consideration, to test whether they are meeting the requirements in a fashion suitable to the customer. When the customer's requirements are reasonably met, they then complete functionality of the software package [16;11-18]. Other models exist, but these two are the ones that are important to us.
Figure 1: The Classic Life Cycle [16;13].

Figure 2: The Prototype Life Cycle [16;16].
Two particularly important steps within the life cycle are analysis and design, methodology paradigms. They are the planning stages that occur before implementation (programming) of a system, and overall system quality depends on their effective and efficient completion. Averting mistakes in these phases greatly reduces error correction time for software, and hence project costs (Figure 3). Several paradigms for accomplishing these steps are around, but two are of particular importance; these methodologies are Structured Analysis and Design (SAD), and Object-Oriented Analysis and Design (OOAD).

It is the purpose of this project to examine a given system with both of these AD methods to see which yields a "better" solution. The system under consideration is the RECLAIM system (See Appendix A). I completed a SAD of this system in the spring term of 1993 as part of a group project. Now, as of the summer of 1993, I have completed an OOAD of the system. Following, I shall give an explanation of the methodologies involved, and compare their strengths and weaknesses, in an attempt to determine which of these two major analysis and design strategies yields a better result.
Figure 3: The costs of errors in the life cycle [10;16].
Methodologies

Of the two methods, Structured Analysis and Design is the older, established AD technique. It is a functional method. "In the functional view a software system consists of data items that represent some information, and a set of procedures that manipulate the data" [10,159]. SAD requires that functions, i.e., processing, are the active elements of the software, whereas data elements are in themselves totally passive "containers of information" [13;136].

The specific SAD approach applied in this project was documented by Edward Yourdon in his books Modern Structured Analysis and Structured Design, which build on Ward and Mellor's methods from Structured Development for Real-Time Systems. This AD technique is a synthesis of the more useful techniques surrounding the data flow diagram (DFD) description method, but more on that later.

The primary goal in SA is to build an Essential Model, a model of what the system is to do to satisfy the user's requirements, disregarding how it will be done. It has two components, the Environmental Model and Behavioral Model. The Environmental Model defines the system against the rest of the world, with consideration for the interfaces, or boundaries, between them. It includes a statement of purpose, a context diagram (a special DFD that shows data flow between non system components and the system), an event list that describes actions that the system is responsible for responding to, a data dictionary (DD), and an entity-
relationship diagram (ERD), which is a network model for describing data at a high level of abstraction. The Behavioral Model describes the system itself, or what it must do functionally, with Data Flow Diagrams (DFDs). In DFDs, named bubbles (circles) represent processes and parallel lines with a name between represent stores. Data flows, or unidirectional named lines with arrows, represent the transfer of data between processes and stores. The first set of DFDs produced describes the bottom level of the system; they correspond to the event list.

Figure 4: A context diagram. The entire system is represented by the bubble [20,339].
Once these models are obtained, they must be reconciled. First, the DFDs and dictionary are put in balance. The DFDs are upward leveled and the context diagram is lower leveled. Leveling means either combining bubbles (processes) and their data flows into one bubble with data flows into and out of it (upward), or taking a bubble and its flows and breaking it down into multiple bubbles and flows (downward). The idea is to do this from the top (context diagram) and bottom (lowest level DFDs), and meet, balancing the successive levels of diagrams. In total, a view of the system is achieved in which deeper and deeper levels of complex behavior can be viewed by looking deeper and deeper into the diagram levels (see figure 6). A last step is to make sure that the DFDs, ERD and DD are complete and balance against each other. Since the system should be the same from any perspective, items in one diagram should not be missing from another; for example a data store in a DFD must appear in the data
dictionary with its definition in order for the models to agree.

Figure 6: A balanced DFD fragment [20;170]
Structured Design takes the DFDs of Structured Analysis and turns them into modules (represented by boxes) with top-down hierarchy. The DFDs are analyzed at the bottom level. They are divided into transform and transaction designs. In transform analysis, the bubbles are divided into afferents, or input bubbles, transform centers, or data processors, and efferents, or output bubbles (Figure 7). The transforms are factored so that the afferent and efferent bubbles become child modules of the transform center; thus top-down hierarchy is created. Transaction design is similar except that a transaction center has few inputs and based on those inputs it calls any one of a number of processor flows to handle the signified action (Figure 8). Here again hierarchy is created by the transaction centers that become the parents in the structure.
Figure 7: A Transform Center [21;193].

Figure 8: A Transaction Center [21;226].
Object-orientation has reached maturity. Virtually all areas of software science and technology have now recognized its significance and effectiveness. Even the COBOL community, one of the most conservative software communities, is now engaged in designing object-oriented COBOL languages [13;v].

With such avid enthusiasm, one would think that object-orientation is the cure for all software projects.

As one might guess, object-orientation is fundamentally different both in how it views the system to be modeled and the procedures of the modeling process. As far as viewing the system, OOAD tries to provide a more concrete attachment to the real world by using "Objects" as the primary building blocks of the model. These Objects provide behaviors that manipulate data that they contain. In this way, real world objects should more easily be mapped onto the Objects of the model [13;136].

The OO model provides both procedural and data abstraction through association of procedures and data exclusively with an object. These differences might not strike the casual observer as great, but they represent a different way of thinking about engineering software. Ed Yourdon, who put in writing the SAD methodologies now widely practiced, has completely converted to the OO philosophy for not only AD but implementation, or programming, as well. The techniques he and Peter Coad developed in the books Object Oriented Analysis and Object Oriented Design are considered
the premiere techniques of the practice. This methodology was used exclusively in the OOAD of RECLAIM presented here.

Figure 10: Object-oriented modeling attempts to capture reality as closely as possible [4;32].

The first step in OOA is to look at the problem domain, or system under consideration, and identify what are called Class-&-Objects. In the sense that the analyst is trying to "match the technical representation of a system more closely to the conceptual view of the real world," Class-&-Objects in the system are approximately the items in the real world that anyone would call objects. There is an extensive list of procedures by which to derive Class-&-Objects, some of which might be less obvious. These include: structures, other systems, devices, things or events remembered, roles played, operational procedures, and sites. Rules are provided by which to challenge the Class-&-Object candidates.

Once Class-&-Objects have been found, structures are identified. Mankind has developed two basic structures over history, the generalization-specialization or gen-spec structure, and the whole-part structure. In gen-spec structures, the Class is the generalization, and Objects that belong to the Class are instantiations, or the
specializations. In this hierarchy, Objects inherit the characteristics of their parents (Attributes and Services), with perhaps modifications for their own inherent needs. Objects may belong to multiple classes, thus affording for greater inheritance and wider specialization. In the whole-part hierarchy, the whole is an Object and the part is another Object that is considered part of, or to belong to, the whole in some quantity. As a last form of hierarchy, in larger systems with many Class-&-Objects, Subjects are created to maintain readability. These subjects help to add scale, allowing for better visibility of the model and its components.

The final steps, which involve most of the work, are to add Attributes and Services to each Class-&-Object. Attributes are the data, or states, needed to understand the Object in question, including how it should behave. When an Object must associate with another Object to fulfill its duties, an Instance Connection is created to relay this information through the model. An Attribute whose values causes fundamental changes in what Services can be performed on an Object requires production of a State Transition Diagram for its Object. Services are the processes that need to be carried out on the Attributes of an Object. They are triggered by messages sent from other Objects. When an analyst wants to show this mapping, he uses a Message Connection. Services are specified in detail in Service Charts. These are the complete steps in OOA (see Appendix
B). It should be noted that they may be carried out in parallel and iteratively as well as sequentially.

Figure 11: A complete OOA for a vehicle registration system [4;170].
OOD is the same process as OOA, except OOA is applied to different components of what is viewed as the entire model. Whereas in strict OOA the Problem Domain Component (PDC) was analyzed, in OOD the Human Interaction Component (HIC), Task Management Component (TMC), and Data Management Components (DMC) are analyzed. The HIC accounts for how humans will interact with the system. The TMC belongs to models where multi-processing, or the execution of multiple tasks, must occur (near) simultaneously. The DMC is "the infrastructure for the storage and retrieval of objects from a data management system;" it isolates data management. The OOD of these components can be carried out as (simultaneous) activities as opposed to sequential steps. OOAD is a seamless AD method because of the sameness of the processes involved in implementing it.

Figure 12: The seamless "multicomponent, multilayer" OOAD model [4;26].
**Results**

Structured Analysis has some inherent strengths in its methodology. SA is based upon the fundamental concept of sequential flow, a concept well understood by computer programmers. This makes modeling with SA a relatively painless task.

Structured Analysis is a complete model. With the various diagrams and the data dictionary, the analyst is less likely to make a mistake. The reason for this is balancing the models against one another has been made a critical component of the system. In practice, my group found this quite effective, especially since we divided the diagrams and dictionaries among us. When we sat down and put them together, we saw that each of us had discovered in our models aspects of the system which others had not in theirs. This checking and balancing brought us much closer to a complete representation of the system.

However, this finding uncovered a deficiency in style which Coad and Yourdon note that SA supports: too much emphasis on the DFDs. The DFDs took by far the most amount of time and effort to complete; however, we had no problem with this because the DFDs actually were the backbone of the representation of the system. What gave us the most trouble was that balancing came after all the models were fully developed. This made it almost instinctive to reject and find fault in the other models when they did not agree. So it was only with great effort that I as the analysis quality
assurance person could persuade that fundamental changes be made to the DFDs.

In some cases the other parts of the SA model had to suffer from problems with the DFD. My primary example for this is what I like to call the "merge syndrome." During the upward leveling process, when the lower level bubbles are merged to get a higher level bubble, we almost invariably had to merge for the inputs and outputs of the grouping. This is because the data flow inputs and outputs for the encompassing bubble the next level up must be balanced with those below in that they must convey the same data. We could not leave the flows unmerged because then all the parent bubbles of the upper level with their data flows intact from below would make a diagram that looked like spaghetti and meatballs. Hence in the data dictionary many definitions can be found that are simply conglomerations of other data dictionary items.

All of this is hinting at the inherent weakness of Structured Analysis, which is its near total lack of regard for data in the system. The tools of SA force the software engineer to concentrate almost exclusively on the necessary processing of the system, what the system must do. Obviously it is not a bad thing to understand what the system should accomplish; rather, it is necessary. However, when a method like SA advocates jumping in and representing the system almost solely with processing, it creates volatility in the resultant model [4;22-30].
My team was easily aware of this. We carefully crafted the Event List, checking and double checking, to make sure that no events were left out. Why? Because we could see that any change in the Event List would cause fundamental changes in the DFDs; in some cases perhaps a complete reworking of all the tedious diagrams we were to produce. We had no problem with this in the class, because Dr. Davis did not change the system specifications on us. But in the real world where the customer is likely to omit system specifications, the results could be devastating to the models of analysts. In my mind, the lesson to be learned here is that analysts spare as much time as possible to encompass their systems.

Structured Design has an exclusive strength: modularity. Its aim is to convert the DFDs of the SA into modules. I know from experience that programming by modules works, and not only does it work, but it is efficient. The difficult part is to get from the bubbles (DFDs) to boxes (modules). In general this cannot be an easy task but I would submit that it is not as insurmountable as Coad and Yourdon would have us believe. In my work group's case, during the analysis phase we had inadvertently modeled the whole RECLAIM system into a transaction form. Though Yourdon warned against such a move in Modern Structured Design, we felt compelled to model the entire system as a transaction. We were then able to proceed with the factoring of the system.

That is, until we ran into a truly insurmountable
problem. We had from the beginning intended to implement our system in Virtual BASIC, a programming environment that greatly automates creating GUI (graphical user interface) systems. After the first factoring, we realized that the parts of the system that would be handled through Visual BASIC could not be separated from the structure chart; these items corresponded in some cases to complete modules but in other cases not. So the Hierarchical Structure Chart and Interface Structure Chart that we developed for RECLAIM do not methodically follow from the previous charts. Coad and Yourdon refer to a magic transition that often takes place from SA to SD because of their inherent irreconcilability; in our case, I now believe that this is not so much the result of changing methods from SA to SD but because in SAD the software engineer is not forced to consider a Human Interaction Component. As we shall see, this is not the case in OOAD.

Comprehension and stability are the great strengths of OOA. One invariably can see the real world reflected in the Class-&-Objects of the system. Names and the fundamental hierarchical relationships that mankind uses to classify objects are preserved. Beyond that, and more important, is the stable model of the system under consideration that the software engineer gains. I related previously that SAD is flawed in that it tries to model the system almost entirely by processing; with this perspective, any changes in required system behavior might wreck the entire lot of DFDs modeling
the system. Take the same change in OOA's perspective. If items are added to or removed from the system, then they are added to or removed from the Class-&-Objects listing. I initially had Objects for maintenance history and maintenance schedule until I realized that they were computed items and not Objects; getting rid of them only involved deleting them from the list. If behavior of an Object changes, change the Services. This localization of the effect of changes in the system on the model is what makes it stable and coherent; the model can weather indecisive requirements and fickle customers better than any method that I have seen.

Unfortunately beyond these two things that encompass Subject, Class-&-Objects, and Structure layers, the model is both incomplete and at times unviable. Attributes present problems in the OOA methodology. Coad and Yourdon insist on atomic attributes. As Ian Graham says,

Under the influence of the relational database movement, they insist that attributes should be atomic. This is wrong. Object-orientation is about modeling complex objects which need not be atomic...[8;233].

Without the ability to have records and arrays in Objects, it is unclear how to continue except to explode the number of Objects in the park, making it unnecessarily large. In the case of Object Park, I included list structures anyway. This example falls under a more general problem category inherent to OOA: when is something an Attribute as opposed to an Object? This is an especially troubling problem when trying
to distinguish between an Attribute and a Whole-Part Structure.

Services create even bigger problems. They are divided into categories of simple and complex. The "simple" Services are create, connect, access, and release; they are implicit OOA charts and are not directly shown (a good way of hiding coupling). No explicit guidance is given on how these Services function in behavior or what to do with them, but yet Coad and Yourdon state that 80-95% of all behavior is handled here in many systems. Hence, in my OOAD of RECLAIM references to these Services are like functional calls for lack of better understanding. Most of it is connecting a part and whole; it is never stated whether this is implicit within the creation of the part, so I had to assume that I was responsible for it.

A simple Service which Coad and Yourdon neglected entirely was the "WhoAmI" service. With it, Objects can be made aware of other Objects. Because I was unable to use this Service, I had to specify Attributes ObjectType, EcoType, EnvType, and ReqType. This made the Object, EcologicalCharacteristic, EnvironmentalFactor, and MaintenanceRequirement Objects "knowable" so that error checking could be performed.

Another major problem I had with Services is that they are only initiated by messages from other Services, but then those are only initiated by messages from other Services... Couple this with the fact that it is unknown whether Services
of the same Object may call each other. Then it is perhaps easier to understand why my OOAD model carries out actions in a top-down manner, wholes calling parts and handling the necessary processing based on the results, often of observations of Attribute values in the parts.

The strength of OOD lies in the fact that Coad and Yourdon take into account the components of the system that, though apart from the PDC, are equally necessary to the functionality of the system. Whereas the RECLAIM structured design fell apart when we factored in the GUI, I had no problems with a RECLAIM GUI in OOD because of the HIC. These components are developed the same as the PDC in OOA. So OOA and OOD are seamless processes, OOD building directly on the result of OOA. In fact, in retrospect I think it better to carry out OOAD simultaneously. This way, the system develops evenly and problems between components can be discovered earlier.

What is the major problem with Coad and Yourdon's OOD? They simply do not tell enough about the HIC, DMC and TMC. Only a highly empathic reader could understand fully what these components are and how to implement their AD when an average of 11 pages, all generalities, are spent on each one. Hence a major problem that one has is how the different components interact. For instance, my OOAD PDC is enslaved to the HIC because in part from the recursive Service syndrome (note that the HIC itself is assumed called up by the System Management facilities), but also because there are
no guidelines on how to interface the components.

All the stated problems with OOAD are indicative of the wider problem with it. It is great at organizing static structures, but poor with dynamics. Dynamics are the processing and data passing parts of the system. Of course, this is what SAD models well. It is fairly clear that these two paradigms are like opposite sides of the coin: the one the information-oriented model, the other the processing-oriented model.
Conclusions

If I were asked to choose between SAD and OOAD for now, then I would choose SAD in a classic life cycle environment. Though it has its problems, most of them can be overcome. The strong emphasis on the DFDs can be tuned down in favor of reiterating the importance of the ERD and DD. The ERD is actually a component part of the OO diagrams, and its emphasis would help stabilize SA. Also, the gap between SA and SD can be overcome; it is not easy but not impossible. Components like the HIC can be incorporated with practice; real-time, critical-thread processing has already been introduced into SAD.

On the other hand, I do see OOAD, probably in a prototyping environment, gradually replacing SAD through its evolution. It is the more competent, stable model. At this point however, terminology and meaning are still too undefined; methodology is not specific enough. Coad and Yourdon's method is regarded as the best OOAD model available, yet next to no helpful information is available in the OOD book. An undefined method leads analysts and designers into pushing details off until implementation (onto someone else's shoulders), a result of which could be a project failure. Even Coad and Yourdon are quick to point out that OOAD is not a silver bullet, and has yet to reach maturation. Through practice and experimentation I believe it will be refined and brought to acceptable standards.

An important point to remember is that design style
tends to derive from the language and analysis style then from the design. So knowing how to program in the regular languages made me familiar with SAD. On the other hand, I have no experience with OOPLs, and therefore the aims of OOAD are all the more mysterious. Some of the problems I encountered with OOAD might be the result of not understanding where the AD results will fit into an implementation scheme. Yet even object-oriented languages differ widely, as there are no concrete standards yet for them to practice.

An important factor then will be the spread of OOPLs in the computer world. SAD is familiar because its aims are those learned and understood by programmers: modules, top-down and bottom-up design, processing orientation; these are the results of the command imperative languages. As languages actually become object-oriented, instead of just claiming it, and institutions such as the universities acquaint future programmers with object oriented features, object orientation will gain not only acceptance but may surpass process orientation as the language of choice. In this case, OOAD will certainly replace SAD, as there is no reasonable way to move from SAD to OOPLs.

In sum, producing large, reliable software packages is a complex process that requires much skill and is still, even with software engineering, mostly an art. No clear paths exist to methods that will guarantee results, nor are they ever likely to. Yet, as everyday we come to rely more and
more on machines and their software for simple as well as critical services, software engineers must continue the process of uncovering new and better methods of software creation, for everyone's sake.
BIBLIOGRAPHY (WORKS CONSULTED)


19. STROUSTRUP, BJARNE. *What is Object-Oriented*


Appendix A
Structured Analysis
ORIGINAL PROBLEM STATEMENT

LITTLE RIVER CANYON NATIONAL PARK MANAGEMENT SYSTEM

The Little River National Recreation Area is a new tract of several thousand acres in the National Park Service System which was created recently by an Act of Congress. Since the national parks have been receiving excessive numbers of visitors, it has been decided that a computer based system should be developed to manage this and similar areas across the country. The recreation area consists of a winding river and surrounding canyon, heavily forested areas, and a large lake plus meadowland. Many of the forested areas are traversed by old logging trails and are maintained for hiking and bicycling.

The product to be produced will be called the Recreation Area Loading and Management Information (RECLAIM) system and its requirements are listed below.

1. The system shall be able to display and store descriptions of various natural parts of the recreation area. A map of the area will be normally divided into 10,000 square foot cells, and ecological characteristics shall be associated with each cell. If a particular cell has diverse ecological characteristics it may be further divided in order to better describe its characteristics. The characteristics include; soil type, type of cover, water table height, and water quality in the lake and streams. Individual objects within a cell such as a lake shall also be able to be identified separately if required.

2. The system shall be able to display and store descriptions of any constructed facilities in the area, including barns, lodges, restrooms, log bridges, lake docks, picnic tables, shelters, and tennis courts.

3. The system shall allow for easy addition of new ecological characteristics and facilities to the database as required.

4. The system shall be able to associate a maintenance requirement for each cell or selected object in the park database. Using this data the system shall generate a maintenance schedule for the entire recreation area.

5. The system shall allow the entering of the effects of the use of the facility by the public on the recreation area environment. This may also be on a cell by cell basis or an object by object basis. This shall enable administrators to best accommodate the many visitors by adjusting traffic levels to minimize damaging the environment.

6. The system shall be user friendly allowing users the ability to easily generate the maintenance schedules, alter the database and determine environmental effects.
MODIFIED SYSTEM SPECIFICATIONS

This modified system specifications listing consists of the original problem statement, altered according to changes made through requirement change forms. Alterations are denoted by underline.

1. The system shall be able to display and store descriptions of various natural parts of the recreation area. A map of the area shall be divided into various cell sizes, each square in shape so there are no gaps. The largest cell will be the entire park. Every cell may be broken down into four smaller square cells, down to a minimum of 10,000 square feet (100 feet on each side). Each cell may have distinct characteristics. The characteristics include; soil type, type of cover, water table height, and water quality in the lake and streams. Individual objects within a cell such as a lake shall also be able to be identified separately if required.

2. The system shall be able to display and store descriptions of any constructed facilities in the area, including barns, lodges, restrooms, log bridges, lake docks, picnic tables, shelters, and tennis courts.

3. The system shall allow for easy addition of up to six new, user-defined ecological characteristics in addition to the four specified in the original problem statement (soil type, type of cover, water table height, and water quality.), for a total of ten.

4. The system shall be able to associate a maintenance requirement for each cell or selected object in the park database. These requirements will consist of cutting grass, cleaning, picking up trash, painting, fertilizing, and raking, plus up to nine user-defined maintenance requirements, for a total of fifteen. Using this data the system shall generate a maintenance schedule for the entire recreation area. Furthermore, a history of maintenance done will be kept by the system for a rolling year. (That is, there will always be one year of data.) This history will be in the form of a list of dates the maintenance requirement was done.

5. The system shall allow the entering of the effects of the use of the facility by the public on the recreation area environment. This may also be on a cell by cell basis or an object by object basis. This shall enable administrators to best accommodate the many visitors by adjusting traffic levels to minimize damaging the environment.

6. The system shall be user friendly allowing users the ability to easily generate the maintenance schedules, alter the database and determine environmental effects.
EVENT LIST

F  1. Request for object display.
F  2. Request for cell display.
F  3. Request for object data.
F  4. Request for ecological data change for cell.
F  5. Request for ecological data change for cell.
F  6. Request for ecological data addition.
F  7. Request for maintenance schedule change for a cell.
F  8. Request for a maintenance schedule for a cell.
F  9. Request for additions to maintenance schedule for cell.
F 10. Request for environmental data for cell.
F 11. Request for environmental data change for cell.
F 12. Request for an addition to environmental data for cell.
F 13. Request for maintenance schedule change for objects.
F 15. Request for additions to maintenance schedule for objects.
F 16. Request for environmental data for an object.
F 17. Request for environmental data change for an object.
F 18. Request for environmental data addition for an object.
F 19. Request for object addition.
F 20. Request for subdivision of cell.
F 21. Request for change object in a cell.
CONTEXT DIAGRAM

User → RECLAIM
  user-input
  terminal-display

RECLAIM → Printer
  maint-schedule
2 Add Item

2.1
Add Item to Object

2.2
Add Item to Cell

- add-object-command
- add-data
- add-cell-command

Cells

- system-message
2.1 Add Item to Object

Diagram:

- **2.1.1** Find Object and Get New Data
  - add-data
  - system-message
  - env-data
  - requested-object
  - main-data

- **2.1.2** Add Environmental Data to Object
  - system-message
  - env-data
  - requested-object

- **2.1.3** Add Maintenance Requirement to Object
  - system-message
  - add-ms-to-object-command

- Cells
2.2 Add Item to Cell
2.2.6 Add Children to Cell
3.2 Change Cell

- Get New Data
  - change-data
  - message

- Change Object in Cell
  - change-object-in-cell-command
  - update-display
  - object-data

- Change Environmental Data of Cell
  - change-env-data-of-cell-command
  - env-data
  - update-display

- Change Ecological Data of Cell
  - change-eco-data-of-cell-command
  - eco-data
  - update-display

- Change Maintenance Schedule of Cell
  - change-ms-of-cell-command
  - update-display
5 Display

5.1 Determine Means

5.2 Manage Terminal Display

5.3 Print Data

output

requested-terminal-display-data

maintenance-schedule-data

terminal-display

maintenance-schedule
5.2 Manage Terminal Display

- **5.2.1 Display Graphics**
  - Input: requested-terminal-display-data
  - Output: graphics-display

- **5.2.2 Display Text**
  - Input: requested-terminal-display-data
  - Output: text-display
DATA DICTIONARY

CONVENTIONS:

*Italic* = control signal
*Bold Italic* = terminal control signal
*Underline* = data-flow
*Bold Underline* = terminal data-flow
+ = logical AND
| = logical OR
() = grouping


(add-command = add-object-command | add-cell-command)

(add-data = (requested-object + (env-data | maint-data)) | object-data | env-data | eco-data | maint-data)

(add-object-command = add-env-data-to-object-command | add-ms-to-object-command)

(cell = the cell-data currently selected by the user for viewing)


(child-reference = reference to a sub-cell by its parent cell)

(Cells = storage for cell-data)

(cell-size = 10,000 sq. ft | 40,000 sq. ft | 160,000 sq. ft | TBD)


(change-command = change-object-command | change-cell-command)

(change-data = (requested-object + (env-data | maint-data)) | object-data | env-data | eco-data | maint-data)

(change-object-command = change-env-data-of-object-command | change-ms-of-object-command)

(command = input that generates (add-command | change-command | retrieve-command))
date = when maintenance requirement was last performed
date = when maintenance requirement was last performed
eco-data = type of cover | soil type | water table height | water quality | [user specified]
env-data = environmental impact data to be specified by user
frequency = how often a maintenance requirement is to be performed
graphics-display = symbolic representation of cell-data
maint-data = (frequency | date) + maintenance requirement + maintenance history
maintenance history = rolling year of previously entered dates
maintenance requirement = clean | cut grass | fertilize | paint | pick up trash | rake | [user defined]
maintenance-schedule = printout of maintenance-schedule-data
maintenance-schedule-data = (object | cell) + maint-data
message = "cell size not divisible" | "object not found" | "update complete" | TBD
object = (barns | lodges | restrooms | log bridges | lake docks | picnic tables | shelters | tennis courts | user specified)
object-data = object | env-data | maint-data | user specified
output = system-message | cell-data
requested-object = object to be changed or retrieved by user
requested-terminal-display-data = cell data | system-message
retrieve-cell-data-command = retrieve-object-in-cell-command |
retrieve-command = retrieve-cell-data-command | retrieve-object-data-command
system-message = update-display | message
terminal-display = graphics-display + text-display
text-display = textual description of cell-data
user-input = command + (add-data | change-data | requested-object)
Structured Design
Original DFD Mapping to Struct. Chart

Selector

Processor

Add

Retrieve

Change

Add Cell Data

Add Object Data

Retrieve Object Data

Retrieve Cell Data

Change Object Data

Change Cell Data

Add Object Data to Cell

Add Environmental Data to Cell

Add Biological Data to Cell

Change Object Data of Cell

Change Environmental Data of Cell

Find Object

Get Data

Determine Means

Print Data

Manage Terminal Display

Display Graphics

Display Text

Design Document
Appendix B
Object Oriented Analysis & Design Notations
Figure 13: Layer Models of OOA/OOD [5].
Figure 14: Class-&-Object Specification Template [5].

```plaintext
specification
  attribute
  attribute
  attribute
  externalInput
  externalOutput
  objectStateDiagram
  additionalConstraints
  notes
  service
  service
  service
  and, as needed,
  traceabilityCodes
  applicableStateCodes
  timeRequirements
  memoryRequirements
```

Figure 15: Object State Diagram Notation [5].

```
[-------------------] State
\               \      Transition
```

Figure 16: Service Chart Notations [5].

```
< ____,> Condition (if; pre-condition; trigger, terminate)
[ ] Text block
[ ] Loop (while; do; repeat; trigger/terminate)
| Connector (connected to the top of the next symbol)
```
Object
Oriented
Analysis
Class-&-Object Layer

- Park
- Map
- Object
- Cell
- Environmental Factor
- Maintenance Requirement
- Ecological Characteristic
Structure Layer

[Diagram showing relationships between Park, Map, Cell, Object, Ecological Characteristic, Environmental Factor, and Maintenance Requirement]
specification Park

attribute ParkName

attribute Location

attribute ObjectList
ObjectList details a list of
• every object type
• the icon for each object type
• a list of maintenance requirements performable on object
• a list of associable environmental factors

attribute EnvList
EnvList is a master list of EnvTypes

attribute EcoList
EcoList is a master list of EcoTypes

attribute MaintList
MaintList is the master list of ReqTypes
service Create

While number of Cell <> 256

Creates a Cell and connects to it

Returns
specification Cell

attribute XCoordinate
XCoordinate is the location of Cell along X axis of park grid.

attribute YCoordinate
YCoordinate is the location of Cell along Y axis of park grid.

attribute CellMap
CellMap is a 16x16 matrix of icons of objects in the cell.
service AddObject (in: Object values, out: result)

For every object in (connected to) this cell

Is the Object's CoordinateList completely covered by CoordinateList of new Object?

yes

Sends message DelObject to delete the old Object

no

Is there overlap between the CoordinateLists of the Objects?

yes

Accesses CoordinateList of old Object and removes shared coordinates

no

Does exact Object exist already?

yes

Adds coordinates to CoordinateList of the existing Object

no

Creates and connects to new Object

Accesses ParkObjectList.Icon

Updates CellMap with Object's Icon

Returns success
service DelObject (in: Object.id, out: result)

Does Default Object (Type Grass, Name NIL) exist?

- no
- yes

Creating default Object

Accesses Object.CoordinateList of Object to be deleted

Gives CoordinateList to default Object

Sends message Object.Release

Returns Success
specification Object

attribute ObjectType
Identifies Object; comes from ObjectList of Park.

attribute ObjectName
What the user calls the object.

attribute ObjectDescription
How the user describes the object.

attribute CoordinateList
The (x,y) coordinates this object occupies in the Cell it belongs to.
service Release

While there is a connected Environmental Factor

 Releases an EnvironmentalFactor based on cid

While there is a connected MaintenanceRequirement

 Releases MaintenanceRequirement based on cid

 Does normal Release
service AddEnv (in: Env data, out: result)

Accesses ObjectList.ObjEnvList

Type of env to add is element of ObjectList.ObjEnvList?

yes
Create Env and Connect to it

Returns success

no

returns "EnviromentalFactor Mismatches Object" failure
service AddReq (in: Req data, out: results)

Accesses ObjectList.ObjReqList

Type of req to add is element of ObjectList.ObjReqList?

yes

Create Env and Connect to it

no

Returns success

returns "Maintenance Requirement Mismatches Object" failure
specification EcologicalCharacteristic

attribute EcoType
The type of ecological characteristic (from EcoList of Map).

attribute EcoInfo
User specified ecological information (text).
specification EnvironmentalFactor

attribute EnvType
What type environmental factor this is (from EnvList of Map).

attribute EnvInfo
User description of environment.
specification MaintenanceRequirement

attribute ReqType
What type of requirement this is (from MaintList in Map).

attribute Frequency
How often to perform the requirement.

attribute BeginDate
The day on which to begin performing the requirement.
Object Oriented Design
NOTES ON THE OBJECT-ORIENTED DESIGN

1. HIC

The author was not able to provide a full development for the HIC of RECLAIM. Limited experience with object-style interfaces in conjunction with time constraints has led to a very simple version of a HIC that is more descriptive than implementative. For instance, windows and menus are used, but attributes and services for these classes are left undefined. All buttons, scroll-bars, etc. are not represented as objects. Also, though the windows for map, ecological, environmental, and maintenance data displays are almost exactly alike in the design, their implementations would actually be quite different, so there is no justification for a class to which they all belong; this problem is a result of the descriptive nature of the HIC models.

The HIC presented here was meant for a Macintosh running Apple's system software. The system software consists of Event, File, Menu, Dialog, Window, Memory, and Resource Managers. In reality, these work in tandem with Mac applications. For the purposes of this design, however, the system software was viewed as a driver.

2. There was no TMC for RECLAIM.

3. There was no DMC implemented for RECLAIM in OOD (or SD).
specification FileMenu

notes
The FileMenu responds to requests for the normal file and printing operations. It depends on File Manager to handle saving files. Other operations such as Page Setup and Printing are assumed to be part of the system's capabilities and as such it just passes on the requests to the system. The assumption is that the active window can be printed through the Print routines of the system.

externalInput
• called by Event Manager
service New

- A RECLAIM window is already open?
  - yes
  - no
    - Opens a dialogbox and informs user only one RECLAIM window is allowed open at one time
    - Opens a dialogbox that returns a ParkName and Location
      - yes
      - no
        - Creates park giving it initial values
    - Returns to the Event Manager
service Open

Calls system to return RECLAIM File

System found a file?

yes

Pops up a RECLAIM Window

no

Makes necessary system calls to File Manager to load the file

Returns to the Event Manager
service Save

Is a Park open?

- yes
  - Calls File Manager to open a file with name of park
  - Saves each object in the park through calls to the File Manager
  - Calls File Manager to close Park
  - Returns to the Event Manager

- no

yes

Is a Park open?

- yes
  - Calls File Manager to open a file with name of park
  - Saves each object in the park through calls to the File Manager
  - Calls File Manager to close Park
  - Returns to the Event Manager

- no
service PageSetup & service Print

Calls standard system routine to handle the service

Returns to the Event Manager
specification EditMenu

notes
The EditMenu responds to the requests for editing items in the park. These include add and delete. It checks for the active window, i.e. the one that the user has most recently clicked, and calls the appropriate routines within for those windows that handle additions and deletions. It reports a failure to the user.

externalInput
• called by Event Manager
service Add

Gets active window from the Window Manager

Is active window the RECLAIMWindow?

yes

accesses MapWindow.ParkLevel

ParkLevel = bottom?

no

no __ _

returns to the Event Manager

no

Error returned?

yes

Informs user with messagebox

no

Selects add service for the active window and sends message

Informs user that he cannot add at this level through a messagebox
service Delete

- Gets active window from the Window Manager

  Is active window the RECLAIMWindow?

    - yes
    - no

  - accesses MapWindow.ParkLevel

    ParkLevel = bottom?

      - yes
      - no

      Selects delete service for the active window
      Informs user that he cannot delete at this level with a messagebox

    Error returned?

      - yes
      - no

      Informs user with messagebox

- Returns to the Event Manager
specification MaintenanceMenu

notes
The user generates maintenance schedules and views maintenance histories through this menu. Maintenance schedules are for the entire park. Once a maintenance schedule has been generated it replaces the maintenance schedule for its particular month and hence a rolling year history is kept provided the user does not change the internal date of that the computer uses.

externalInput
• called by Event Manager
service GenMaintSchedule

Opens an inactive window titled Maintenance Schedule

For every Cell in the Park

For every Object in the Cell

For every maintenance schedule that the object has

Calculates what days of the current month that the MaintenanceSchedule falls on

Prints report to the Maintenance Schedule window

Saves window contents to this month's maintenance history file

activates window

Returns to the Event Manager
service ViewMaintHistory

1. Opens a dialogue box to get a valid month
2. Pops up an inactive window titled month+" Maintenance History"
3. Calls system to open file for that month
4. Calls system to read file contents into the window
5. Activates window as sole active window
6. Returns to the Event Manager
specification RECLAIMWindow

notes
The RECLAIMWindow is the full screen window. It contains a pop-up window that has as its selections ObjectTypes, ObjectRequirement, ObjectEnvFactor, EnvList, EcoList, and MaintList. This pop up menu will determine what action to take when Add or Delete is requested on the RECLAIMWindow. RECLAIMWindow also contains quadrant buttons used to navigate down into the park, scroll buttons to navigate at the cell level, an up button to move up a level in park view, and a park button to jump to the top of the park (see entire park). It also has a text display window directly underneath the MapWindow to allow for display of various data, especially Object data.
service AddtoParkList (out: result)

Opens dialogbox based on the item selected in the pop-up menu

Dialogbox returns values?

yes

Puts new information in the proper Park list

no

Returns success
service DelfromParkList (out: result)

Opens dialogbox based on the item selected in the pop-up menu

Dialogbox returns values?

yes

Removes information from the proper Park list

no

Returns success
service QuadButtonClicked

Accesses MapWindow.ParkLevel

ParkLevel = bottom?

<table>
<thead>
<tr>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Message MapWindow.Down with direction based on which quadrant button clicked</td>
<td>Calls system to &quot;beep&quot; the user to remind that he is at the bottom level</td>
</tr>
</tbody>
</table>

Returns to Event Manager
service UpButtonClicked

Accesses MapWindow.ParkLevel

ParkLevel = top?

Send Message MapWindow.Up

Returns to Event Manager

Calls system to "beep" the user to remind that he is at the top level
service ScrollButtonClicked

Accesses MapWindow.ParkLevel

ParkLevel = bottom?

yes

Sends message MapWindow.Down
with direction based on which
scroll button clicked

no

Calls system to "beep" the user to
remind that he is at the bottom level

Returns to Event Manager
service ParkButtonClicked

Option Button held during click?

- no
  - Accesses MapWindow.ParkLevel

- yes
  - Sends Message MapWindow.Park

While ParkLevel <> top

- Sends message MapWindow.Up
- Accesses MapWindow.ParkLevel

Returns to Event Manager
service WriteText (in: Text)

Calls system to print Text to TextBox of RECLAIMWindow

Returns
specification MapWindow

attribute ParkLevel
ParkLevel keeps track of what level of the park the user is viewing in MapWindow.

attribute CurrentCell
CurrentCell is either the upper-right-hand Cell used to build the current display or else it is the Cell shown in the display when at the bottom, or individual cell, level of the park.

attribute PreviousCellStack
Holds a list of previous CurrentCells so that it is possible to backtrack level by level all the way back to the top of the park.

attribute CurrentObject
The object the user has indicated for manipulation.

attribute ObjectRef
The information necessary to reference an object in a Cell

notes
MapWindow is a smaller window that occupies the upper-left-hand corner of RECLAIM Window. Its title is the
name of the Park. It displays a 16x16 grid of the icons associated with park objects based on park level and quadrant selections of the user.
service MapClicked

accepts information on what was clicked from the system

ObjectRef <> NIL?

yes

Updates CurrentObject by comparing ObjRef against where system reports MapWindow was clicked

Highlights the icons in grid corresponding to CurrentObject

Accesses CurrentObject's ObjectType, ObjectName, and ObjectDescription

Sends message RECLAIMWindow WriteText, passing the data

Sends message EnvironmentalWindow.EnvBuild

Sends message MaintenanceWindow.MaintBuild

no

Returns to Event Manager
service Rebuild

Rebuilds park (uses algorithm from Structured A&D results)

ParkLevel = bottom?

yes

Builds ObjectReference based on CoordinateLists of Objects in CurrentCell

no

Sets CurrentObject and ObjectRef to NIL

Sends message EcologicalWindow.EcoClear

Sends message EnvironmentalWindow.EnvClear

Sends message MaintenanceWindow.MaintClear

Returns
service Down

- Updates ParkLevel, CurrentCell, and PreviousLevelStack based on quadrant selected
- Calls Rebuild
- Returns

service Up

- Updates ParkLevel, CurrentCell, and PreviousLevelStack
- Calls Rebuild
- Returns

service Park

- Updates ParkLevel, CurrentCell, and PreviousLevelStack
- Calls Rebuild
- Returns

service Move

- Updates ParkLevel, CurrentCell, and PreviousLevelStack
- Calls Rebuild
- Returns
service ObjAdd (out: result)

1. Opens dialogbox and gets Object information

2. Sends message ReclaimWindow.WriteText "Select Cells for Object, then Press Return"

3. Performs necessary calls to System to allow and track selection of grids in MapWindow on an individual basis

4. Takes coordinates returned from system and sends message Cell.AddObject

5. Returns results returned to it
service ObjDel(out: result)

Sends message Cell.DelObj

Returns result returned to it
**specification EcologicalWindow**

**attribute EcoRef**

EcoRef tracks Ecological displayed in the EcologicalWindow.

**attribute ChosenEco**

ChosenEco is the EcologicalCharacteristic displayed in EcologicalWindow which the user has chosen, when defined.

**notes**

EcologicalWindow is placed at the lower left-hand corner inside the RECLAIMWindow. When the park is viewed at the cell level, it will display the EcologicalCharacteristics associated with that particular cell. The user may choose an ecological characteristic and delete it, or choose to add one, by clicking the window to make it active, then selecting Add/Delete from EditMenu.
service EcoHighlightClick

Accepts Highlighted line numbers from Window Manager

EcoRef <> NIL

yes

Cross references Ecological Characteristic from EcoRef using first highlighted line

Sets ChosenEco to the selected EcologicalCharacteristic

Calls Window Manager to highlight just that ecological characteristic's listing in EcologicalWindow

no

Returns to Event Manager
service EcoAdd (out: result)

- Pops up a dialog box that obtains the EcoType and the Ecolinfo
- Accesses MapWindow.CurrentCell
- Creates EcologicalCharacteristic and connects to CurrentCell
- Returns success
service EcoDel (out: result)

Yes

ChosenEco = NIL?

Returns failure

No

Sends destroy to ChosenEco

Returns success

---

service EcoClear

Sets EcoRef to NIL

Sets ChosenEco to NIL

Calls Window Manager to clear EcologicalWindow

Returns success
service EcoBuild

Accesses MapWindow.CurrentCell

For each EcologicalCharacteristic of Current Cell

Accesses EcologicalCharacteristic.EcoType and EcologicalCharacteristic.EcoInfo

Calls Window Manager to print them

References them by location in EcologicalWindow in EcoRef

Returns
specification EnvironmentalWindow

attribute EnvRef
EnvRef tracks EnvironmentalFactors displayed in the EnvironmentalWindow.

attribute ChosenEnv
ChosenEnv is the EnvironmentalFactor displayed in EnvironmentalWindow which the user has chosen, when defined.

notes
EnvironmentalWindow is placed at the upper right-hand corner inside the RECLAIMWindow. When the park is viewed at the cell level and a particular object has been selected, it will display the EnvironmentalFactors associated with that particular object. The user may choose an environmental factor and delete it, or choose to add one, by clicking the window to make it active, then selecting Add/Delete in the EditMenu.
service EnvHighlightClick

Accepts Highlighted line numbers from Window Manager

yes

EnvRef <> NIL

Cross references Environmental Factor from EnvRef using first highlighted line

Sets ChosenEnv to the selected EnvironmentalFactor

Calls Window Manager to highlight just that environmental factor's listing in EnvironmentalWindow

no

Returns to Event Manager
service EnvAdd (out: result)

- Pops up a dialog box that obtains the EnvType and the EnvInfo
- Accesses MapWindow.CurrentObject
- Sends message Object.AddEnv
- Returns result returned to it
service EnvDel (out: result)

ChosenEnv = NIL?

yes

Returns failure

no

Sends destroy to ChosenEnv

Returns success

service EnvClear

Sets EnvRef to NIL

Sets ChosenEnv to NIL

Calls Window Manager to clear EnvironmentalWindow

Returns
service EnvBuild

Accesses MapWindow.CurrentCell

For each Environmental Factor of Current Cell

Accesses EnvironmentalFactor.EnvType and EnvironmentalFactor.EnvInfo

Calls Window Manager to print them

References them by location in EnvironmentalWindow in EnvRef

Returns
specification MaintenanceWindow

attribute MaintRef
MaintRef tracks MaintenanceRequirements displayed in the MaintenanceWindow.

attribute ChosenMaint
ChosenMaint is the MaintenanceRequirement displayed in MaintenanceWindow which the user has chosen, when defined.

notes
MaintenanceWindow is placed at the lower right-hand corner inside the RECLAIMWindow. When the park is viewed at the cell level and an object has been selected, it will display the MaintenanceRequirements associated with that particular Object. The user may choose a maintenance requirement and delete it, or choose to add one, by clicking the window to make it active then selecting Add/Delete in the EditMenu.
service MaintHighlightClick

Accepts Highlighted line numbers from Window Manager

MaintRef <> NIL

yes

Cross references Maintenance Requirement from MaintRef using first highlighted line

Sets ChosenMaint to the selected MaintenanceRequirement

no

Calls Window Manager to highlight just that maintenance requirement's listing in MaintenanceWindow

Returns to Event Manager
service MaintAdd (out: result)

- Pops up a dialog box that obtains the ReqType, Frequency, and BeginDate
- Accesses MapWindow.CurrentObject
- Sends message Object.AddReq
- Returns success
service MaintDel

ChosenMaint = NIL?

yes

Returns failure

no

Sends destroy to ChosenMaint

Returns success

service MaintClear

Sets MaintRef to NIL

Sets ChosenMaint to NIL

Calls Window Manager to clear MaintenanceWindow

Returns
service MaintBuild

Accesses MapWindow.CurrentCell

For each MaintenanceRequirement of the CurrentCell

Accesses MaintenanceRequirement.ReqType
MaintenanceRequirement.Frequency, and
MaintenanceRequirement.BeginDate

Calls Window Manager to print them

References them by location in MaintenanceWindow in MaintRef

Returns