Inter-Office Memorandum

To	PE Group (c: Bob Taylor)	Date	June 19, 1978
From	P. Deutsch (as scribe)	Location	Palo Alto
Subject	Minutes of today's meeting	Organization	CSL

XEROX

Filed on: <Deutsch>pe-minutes-6-19-78.memo on Maxc1

This was the first meeting of a working group to produce a draft report on programming environments. The group consists of:

Peter Deutsch
Jim Horning
Butler Lampson
Jim Morris
Ed Satterthwaite
Warren Teitelman

Agenda & structure

We agreed to accept Bob Taylor's charter and Jim Horning's first-order report outline, amended as follows:

> We feel it is important to include good ideas not present in the 3 primary systems -- we don't think there will be very many of them. (A new heading between III.C and III.D, plus some appropriate revision of I.A.)
>
> We must enumerate requirements (V.A) before we can evaluate deficiencies in them (V.B), so V.A needs to be a priority item as well as V.B.

We agreed to begin as Jim H. suggested, namely, to put together a catalog of current good ideas, assign weights to them, and then evaluate our current systems with respect to them.

> We saw three ways to evaluate features: by imagination ("I think X would be a good idea"), from experience ("I think X has been a good idea"), and by measurement ("In real life, people use X a lot"). Since we do not have or expect others to invest the resources required for measurement, we will not attempt to answer questions on this basis.

We obtained commitments from people as follows:

> Deutsch, Horning, and Lampson are fully committed to the group, at least until Horning leaves in roughly 4 weeks.
>
> Morris has some reservations about the charter of the group, but is willing to participate.
>
> Satterthwaite has heavy demands on his time right now but will participate at least as an observer and Mesa wizard.

Teitelman has other projects with demanding time constraints but will participate as well as possible.

We decided on a regular meeting time of 1:30-3:30 PM on Mondays and Thursdays.

Homework for this Thursday's meeting consists of adding to the catalog anything we feel was omitted, and assigning to each item one of three importance levels (essential, useful, marginal) and one of two difficulty levels (easy, hard). It is important to distinguish "intrinsically or structurally hard" from "requiring a lot of straightforward coding".

After taking care of the above, we began discussion of the ideas catalog, starting from a list which was brought to the meeting (file [Maxc1]<Deutsch>pe-ideas-6-19-78.memo).

Discussion of the catalog

We observed that some kinds of tools require more pre-planning than others, e.g. on-line access to documentation can be added after the fact much more easily than a source-language debugger.

We made several small comments on and additions to the list as proposed:

User access to underlying system mechanisms was not originally on the list, and has turned out to be very important in Interlisp.

"Low overhead" for compilation/interpretation at run time may mean something like <1 second if it only refers to type-in, but means something quite different if programs routinely compose and execute other programs (e.g. Lisp EVAL).

Prettyprinting is not just a source-to-source process: it must take some internal, program-manipulatable program representation and produce source from it.

Most of our subsequent discussion centered around the first four items on the proposed list, and in particular the question of how programs should be able to deal with large, somewhat structured data collections.

Addressing and access

We observed that 2^20 items was not a "large" address space, and considered two alternative meanings of "large".

An address space of, say, 2^24 items would be adequate to hold all the code actually being used even in a large system, but not adequate for all of its data (e.g. the American Heritage dictionary).

An address space of, say 2^48 items *would* be adequate for all the code and data of a very large, even multi-machine system.

We tended to favor the former, more conservative definition, since we did not understand how to provide an efficient, robust implementation of the latter (more on the robustness question below). However, we did feel that it was essential that a transition across the 2^24-object boundary not require the kind of wholesale reorganization of programs that such a transition requires in current language systems.

System facilities for accessing large, external data bases are required for several reasons:

Many questions of organization and efficient implementation can be solved once, rather than over and over again by applications.

The system itself needs data base facilities for tools like the librarian.

Access to externally stored data objects needs to be smooth for the debugger and other system facilities, not just the application programs.

Programs normally refer to internal objects with individual references, but to external data bases with both individual references and "mass" queries. We then observed that there are three basic techniques for speeding up references to external data:

Caches (good for individual references);

Using sequentiality properties (good for searching);

Inversion (good for searching).

Integrity

We observed that a programming environment in which the file system is viewed as an extension of the address space must be extremely robust -- much more so than any programming system we now have. We observed that our current practice is to make the "truth" for a data base be a text file, and not expend a lot of effort on armoring the binary version against all imaginable errors. Notable exceptions are the basic file facilities like IFS; we believed that a system that provides robustness facilities usable at higher levels would have a lot of advantages.

Long-term integrity seems to require some kind of history or redundancy information.

To protect against "cosmic rays", it is sufficient to record this information in a form that only the implementing program understands.

To provide Undo capability, the history must be in a form which makes sense to clients.

Another kind of integrity has to do with not losing information. We mentioned the following kinds of ideas in this connection:

A computed object (e.g. a Mesa configuration) should track of how it got made, in enough detail to make it again. The description of the putting-together process should be program-manipulable.

Files should be self-identifying in a way that does not depend on their transfer between storage media or locations.

It should not be possible to destroy all traces of a file containing input (as opposed to purely computed) information. The space requirements can be kept under control by storing descriptions of files (such as changes from other files) rather than all the bits.

We also observed that the dangling reference problem and various garbage collection approaches could be viewed as integrity questions.

Inter-Office Memorandum

To	PE group	Date	June 26, 1978
From	B. W. Lampson	Location	Palo Alto
Subject	Minutes of June 22 meeting	Organization	CSL

XEROX

Filed on: <Lampson>PE6-22.memo

We spent the entire meeting discussing items 5-10 on Peter's newly numbered list (<Deutsch>pe-eval-6-21-78.memo). The following conclusions were reached:

(5) Minimal support for interrupting the program is A. Good facilities are B. There seemed to be no strong religious feelings about the form of a process mechanism.

(6) There was a long and inconclusive discussion. Apparently we don't really know what this point is about. At one extreme of "data trapping" there is a simple address trap, as on early machines. At the other extreme is KRL. No one was willing to espouse either extreme. It was noted that

> programming with data abstractions can help with this problem, since there is then a better handle on when data is being changed.
>
> checking some predicate at preiodic intervals (e.g. on every control transfer) may be quite adequate when data trapping is being used to catch "core smashing" types of bugs. Mesa already has this facility.
>
> many interesting cases can probably be handled by using the primitive trapping facilities of the mapping hardware.

It was agreed that this should be moved to C pending better understanding of what it is all about.

(7) Certain things are required: something to unwind the stack, some way of catching an unwind, and program control of error processing. What should be provided beyond this is not agreed - the controversies raging around the Mesa signal facilities are a reflection of our poor understanding of the problem. "Good" exceptional condition handling is C-333.

(8) It was agreed that access to BitBlt, disk and ethernet should be moved to packages, with priorities A, B and C respectively. "Adequate" access to packed data remains A, "Good" access (i.e. first class citizenship) is B. There was a long discussion of why it is hard to add packed data to Alto Lisp, which centered around a protection question: how carefully should the system prevent the user from smashing its underlying data structures. There wasn't much agreement on this point, but it does seem to have considerable practical importance, since a highly restrictive attitude makes it difficult to code low-level parts of the system in itself.

(9-10) These are closely related. A long discussion led to the conclusion that all this information is currently available in Mesa, modulo the question of how stable the compiler's internal representation of the program as a tree should be expected to be. Straightforward methods (like the ones used in Clisp) will allow compiled code to be attached to source in a user-created data structure, although it is certainly easier to do this in Lisp, where interpretation is simple. Not explored was the usefulness of an interpreter for a subset of the language, such as currently exists in

the Mesa debugger.

It was agreed to continue along the same path at the next meeting.

Inter-Office Memorandum

To	All	Date	July 21, 1978
From	P. Deutsch	Location	Palo Alto
Subject	Minutes of PE group 6-26 meeting	Organization	CSL

XEROX

The file containing the minutes of this meeting was unaccountably lost. See any member of the PE working group (Deutsch, Horning, Lampson, Morris, Satterthwaite, Teitelman) for a hardcopy.

Memorandum

To	PE Group	Date	June 29, 1978
From	Jim Horning	Location	Palo Alto
Subject	PE Minutes, June 29, 1978	Organization	CSL

XEROX

Filed on: [Ivy]<Horning>June29-minutes.memo

Homework:

Peter has discussed the "default" future for Smalltalk. Things that are likely to happen (in addition to anything done by a Programming Environment project) are:

Development of a constraints interface, following Alan Bornings work.

Simplification of the user interface, following Adele Goldberg's ideas.

A major effort to fit the system into small machines, specifically Notetaker.

Information retrieval capabilities.

More work on animation.

We concluded that this work was pretty much orthogonal to our Priority A items; if we want a better Smalltalk environment for CSL, we'll have to do it ourselves.

At Peter's request, Danny Bobrow prepared a discussion of directions in which the Lisp environment will probably evolve if there is no special PE effort. At least some attention will be paid to several of our priority items, so things aren't quite as orthogonal as with Smalltalk, but neither will it be anything close to what we have been hoping for and discussing. Danny also made a special plea to raise the priority of "ability to create fully integrated local sublanguages," which he considers a critical part of the present Lisp environment. For further discussion of this point, see item B3 below.

Butler wasn't present, but has agreed to prepare a similar projection for Mesa upon his return.

Peter reported that he was making substantial progress in integrating the discussion from previous meetings, as reported in the minutes, into an outline based on his "catalog" of June 21. He will circulate at least a partial draft prior to our next (July 6) meeting.

Jim H. distributed the following possible outline for the core of our report (i.e., Peter's integration, reported above):

"Bare Catalog" as a table of contents and coordinate system for what follows.

Amplification and explanation of catalog items.

Analysis of importance of items for E(PE) and/or (EP)E.

Relation to existing environments: analysis of difficulty for Priority A items in each environment; projections for non-EPE development.

Priorities (A vs. A+).

There was some discussion about whether the matrix of catalog items versus concerns should be presented in row-major or column-major order. An omitted concern was noted: on which items is there substantial agreement on the solution, as well as on the importance? The concern for priorities led to some inconclusive discussion of how to define them, and of chickens and eggs; the consensus was to postpone any attempt at further partitioning Priority A.

Discussion of Catalog:

We resumed the discussion of Peter's Priority A items in the *Virtual machine/programming language* category.

A(13). Simple, unambiguous syntax.

Peter: It is important to have syntax equations (or their equivalent) clearly defining a language that can be used safely; this doesn't exist for CLISP, and for Mesa it runs to five densely-packed pages.

Warren: It was more important in CLISP to respond reasonably to a wide variety of errors than to precisely define what the user could do safely.

Peter: I seriously doubt the utility of a system where the gap between what can be precisely defined and what the user can reasonably do is so large.

Jim H: Although backward compatibility was important for CLISP, and a source of significant difficulty, it is something that we should probably be prepared to sacrifice in a new programming environment if it conflicts strongly with high-priority goals.

Miscellaneous discussion finally led to the conclusion that neither Mesa nor CLISP syntax were the source of sufficient problems for the practicing programmer that we should make this a high-priority item. Dropped to Priority B.

A(14-15). Encapsulation/protection mechanisms (scopes, classes, import/export rules); abstraction mechanisms, explicit notion of "interface."

Peter: An interface should be viewed as at least a partial specification.

Jim H: One of the problems with Mesa is that user-defined abstractions (packages) are not first-class citizens in the sense that they cannot be used wherever language primitives can. In building multi-layer systems using packages, it seems important that the user of a facility not need to care whether it is primitive or user-defined. Having said that, let me warn that this is an open research area, and it is easy to be led astray (modules as data types were the source of a significant fraction of the problems with Euclid).

Alan: A crucial aspect of an (EP)E will be the ability to gain efficiency by wiping out or obscuring interfaces and otherwise destroying structure.

Several: This is an important optimization issue for compilers, etc., but is relatively well-understood compared to most of our Priority A items.

Jim H: I think that at the present time we dare not attempt anything much beyond Mesa for (15), but we might model (14) somewhat more closely on Euclid. (Groan from Warren.) Import/export lists provide useful documentation, even if they are automatically derived (cf. our earlier discussion of declarations).

Peter: Control of exported names has proved to be even more crucial than importation; even the Lisp compiler has had to make some provision for it. I rate export control Priority A, import control Priority B.

Ed: Some future version of Mesa (6 or later) will probably take steps in this direction, anyhow.

A(16). Non-hierarchical control (coroutines, backtracking).

Peter: All three systems provide adequate solutions for coroutines, although each leaves something to be desired in cleanliness, generality, and/or efficiency. Only Lisp does backtracking, but this isn't too important, and Lisp probably doesn't do it right, anyhow.

Jim M: Would this category include procedure closures?

Peter: It certainly should!

Ed: The Mesa compiler would be cleaner with closures, it currently gets much the same effect by nesting procedures.

Some discussion of whether such a general mechanism was needed, or whether specialized mechanisms (e.g., generators) would span the more common and important uses. The efficiency issue was raised; provision of automatic object management (garbage collection) may reduce the marginal overhead of allowing for closures.

Peter: Closure is Priority B.

A(17). Adequate run-time efficiency.

Peter: The following rough estimates indicate the general situation:

> Current Smalltalk is a factor of 100 slower than Mesa.
>
> The re-engineered Smalltalk would be a factor of 5-10 slower than Mesa.
>
> Lisp will be a factor of 3-6 slower than Mesa.

If Dorado is a factor of five faster than the Alto, and Mesa efficiency on the Alto is assumed to be acceptable, then just switching to the Dorado should at least buy a couple of years for either re-engineered Smalltalk or Lisp. Thus, to first order, we are OK here.

B(3). Ability to create fully integrated local sublanguages.

[Integrated from discussions that took place at two separate times in the meeting.]

Peter: Danny has spoken to me several times on this--he feels it's an absolutely crucial feature of the Lisp environment.

Jim M: Why has the Lisp environment been so much more successful than any of the others at such integration?

In the discussion, the following factors were mentioned:

> A major enabling feature is the standard internal representation for parse trees in all sublanguages (S-expressions).
>
> There is a completely standard method of sharing names and passing environments.
>
> Lots of system hooks are available to the sublanguage implementor, allowing him to dig his way in incrementally (although pioneers sometimes encounter quicksand).

Warren: An important principle is that the amount of work for a language change should be proportional to the size of the change.

Peter: Some of the things that have made Lisp sublanguages easy are going to be needed for the other language environments, anyhow. For example, the Mesa debugger needs many of these things; a re-engineered Smalltalk compiler could take advantage of them. The essential notion is that a *name scope* must become a first-class object within the language.

Jim M: One of the reasons is the availability of packages for dealing with language-like structures, e.g., prettyprinters and program input.

Warren: Don't ever underestimate the importance of convenient I/O in the interactive environment.

Peter and Alan: One omission by oversight from the Priority A list is a reasonable multi-language interface.

It was agreed that if this meant the ability for programs in each language to call packages in the other two, this would be a 333 item. However, we could probably settle for somewhat less, namely, the ability to drop from Lisp or Smalltalk into Mesa subroutines, as a "machine code" substitute. The more general facility can be Priority B.

Alan: Lisp makes it easy to "go up" by building new systems/sublanguages on top of old. The EPE should also make it easy to "go down" by re-engineering a high-level system to rely on less support mechanism or run on a smaller machine.

Several: This is hard; no generally useful mechanisms are known.

At this point we turned to Tools, and discussed only A(1). Fast turnaround for minor program changes.

Peter: I have proposed to separate this from my original item "integrated editor," since if the turnaround is fast enough, I don't care how it is achieved.

Alan: If the turnaround is fast enough, you *have* an integrated editor.

Peter and Warren: No, by integrated, we mean that the editor knows about the structure of the language system and conversely; an integrated editor can be called from within a program, and can itself invoke programs.

Warren: It is important to be able to apply the full power of the language to whatever problem you have in hand, even if that is a text manipulation problem, and conversely.

Peter: This suggests that the editor should be one of the sublanguages that we have been discussing. However, the interfaces to our more popular editors don't look very much like any of our programming languages.

Warren: Yes, this is a hard research problem that I'm interested in working on: How should we incorporate selections and mouse clicks into our programming languages?

Peter: I propose that we assign Priority A to fast turnaround, and to an editor package that is callable from the language, and assign Priority B to the integrated editor.

Warren: I'm serious. The integrated editor is very high on my priority list.

We agreed to resume discussion on this point at our next meeting, July 6.

c: R. Taylor
A. Perlis

Inter-Office Memorandum

To	PE Group	Date	July 10, 1978
From	P. Deutsch	Location	Palo Alto
Subject	Minutes of 7/7 meeting	Organization	CSL

XEROX

Filed on: <Deutsch>pe-minutes-7-7.memo on Maxc1

Warren was absent from this meeting; Alan Perlis attended.

Agenda, outline & homework

Homework

In order to make our priorities crisper, each of us will take 100 votes and allocate them among features for next meeting. Jim H. originally proposed that the units be "man-months" rather than "votes"; after discussion, we decided on votes so as to avoid tying together the *desirability* of a feature and its *difficulty*. Both the mean and the variance of the allocations are of interest!

We discussed the difference between the two parsings of the label EPE, and decided it was only one of emphasis. Jim H. suggested that a useful way to describe the difference was in terms of what questions we were interested in asking (and answering):

> (EP)E implies an environment for writing programs in which we are more interested in asking questions about the functioning of those programs (but are still interested in developing the environment as such).
>
> E(PE) implies an environment in which we are more interested in asking questions about the environment itself (but must still provide enough features and stability for people to use it to produce real programs).

Anyone who feels this difference affects the priorities of features is free to submit two different allocations of votes. We agreed that we should try to be more sensitive to this difference in our remaining discussion of individual features.

Our current working document does not address two major areas that we want to cover in the final report: distinguishing fundamental features from those which could be added later with relatively little difficulty, and identifying which features enable other features. Jim H. agreed to write something in these areas by the end of next week.

The working document contains two sets of difficulty estimates (Peter's and Butler's), which disagree substantially in some cases. Peter and Butler agreed to work out these conflicts between themselves. Input from other group members is also solicited. We hope to limit group time for discussing these estimates to cases where conflict absolutely cannot be resolved off-line.

Butler still owes the group a document on the probable future course of Mesa, similar to Peter's comments on Smalltalk and Danny's memo on Lisp.

Introduction

Jim H. said he would write an introduction for the report, dealing in particular with the question of how one decides whether one PE is better than another. He suggested there were really only two criteria for judging PEs and PE features:

> whether they improve the economics of programming -- the resources (machine and human) required for the programming process;
>
> whether they improve the quality of the programs produced -- the performance, modifiability, maintainability, etc.

Subsequent discussion led to agreement on these criteria, but also to the observation that since in practice no one used them to measure systems quantitatively, the criterion actually used for evaluation was usually how well experienced programmers liked the system (which, presumably, often led to the same conclusions). We concluded that it was still important to include these criteria in the report, as a qualitative statement.

A different view

Alan gave a somewhat different view of how one should look at PE features. He suggested there were three key notions:

> "closure" - the ability to take any capability (package, tool, language construct, ...) and use it in any context where it made sense semantically;
>
> "linking" - the possibility of taking capabilities developed as experiments and integrating them into a standard base environment;
>
> "editing" - the ability to alter any or all parts of a system easily.

Ideally, a PE designed for experimentation should leave every conceivable binding potentially alterable. Alan's prime example was taking a cut-down form of Mesa and experimenting with adding garbage collection. L* reflects this philosophy -- all levels of the system, from the most sophisticated external language to the most primitive machine-like level, are available to the programmer.

Subsequent discussion revealed little support for this view. We observed that an approach which encouraged people to build their own variants of systems or system subsets tended to destroy the ability to share the results, although it might permit more alternatives to be tried. With respect to Alan's example, we observed that the experiment in itself was very difficult, and that designing an environment to make experiments of that magnitude easy was beyond what any of us knew how to do. Most of us felt that system design almost inevitably required making more decisions at design time than Alan was advocating. Even though Alan's desiderata were a useful way to think about research areas in PEs, they did not seem concrete or attainable enough to incorporate in our present effort.

Language integration

Point (VM/PL) B3, fully integrated sublanguages, took up the rest of the meeting. There were two major threads to the discussion: why Lisp seems so much more hospitable than Mesa to the creation of integrated sublanguages, and what some alternative meanings of "integrated" might be. [I have taken the liberty of rearranging the discussion somewhat in light of what I think we came to understand in the course of it.]

We observed two characteristics of Lisp that made it easy to create sublanguages that could communicate among each other: the existence of S-expressions as a common, simple representation

of programs, and the existence of a single, very simple name environment (atoms) that all sublanguages shared. (This has both advantages and drawbacks: it leads to the "FLG" phenomenon, for example.) The sublanguages that don't take advantage of these characteristics, such as CLISP, QLISP, and KRL, find their lives a lot more difficult. If we were willing to limit the complexity of sublanguages to that of S-expressions, i.e. procedures and conditionals, then we could devise an S-expression-like representation for Mesa also. [This is a bit of a handwave: in particular, it doesn't allow embedding of a reasonable subset of Mesa itself in a sublanguage, and it doesn't address the point that some of the highest payoff comes from the integration of languages, like KRL, that *don't* look like S-expressions.] We also noted that no matter what features the language and environment provided, proper proceduralization of facilities was essential: even Lisp has sublanguages -- in particular, the compiler control sublanguage -- which are implemented so as to interact with the user directly, and which therefore cannot be considered integrated.

To sharpen our ideas about what integration means, we considered a "straw man": a system in which all languages (editor, interpreter, ...) shared a screen interface (window manager) but were otherwise entirely separate. This led us to the following observations:

> Butler put forward this model as one that imposed minimal requirements on the individual subsystems, at least if they only dealt with the screen as a sequential character I/O device. Peter disagreed, observing that despite numerous attempts, no such package had ever been developed for the Alto, and suggesting that this was because no entirely satisfactory model for the interface had been developed. We agreed that things become more complex as the subsystem's view of the display becomes more sophisticated (e.g. an editable document, a bitmap).
>
> While this model allowed for considerable communication (by human transfer of characters output in one window as input to another), it had two serious deficiencies: it made no provisions for communication under program, rather than manual, control, and it required that all transmitted information be in string form. [Note that even transmission of file names requires integration in the sense of sharing a common file system.]

While we did not reach any clear conclusions, we were able to agree on the following:

> This is an important area for discussion, but it needs more time than we have available to us.
>
> Integration really means a common model for communication.
>
> The one catalog entry we now have should be broken down into multiple features. Some of these should be priority A, some B. [We did not address the question of how we would do this.]
>
> If all the other priority A features in the catalog were provided, Mesa could readily support sublanguages of the complexity of S-expressions.
>
> Adequate proceduralization is necessary for a package implementing a sublanguage to be usable.
>
> A major source of difficulty is the sharing or passing of environment information between sublanguages. One part of this difficulty is simply addressing or naming objects to be shared. Another part is making sure that shared objects are interpreted the same way (in Mesa, making sure the communicants share the same declarations for the objects). One way around this is to have a limited number of globally agreed-upon structures, such as strings or S-expressions, and then encoding more specialized languages within them and interpreting them by convention or agreement (as Lisp does).

Inter-Office Memorandum

To	Distribution	Date	July 11, 1978
From	B. W. Lampson	Location	Palo Alto
Subject	Minutes of 7/10/78 EPE meeting	Organization	CSL

XEROX

Filed on: <Lampson>epe7-10.memo

It was agreed that we will discuss Horning's assignment of fundamental/intermediate/add-on attributes to the catalog at the Wednesday meeting.

Rationale: Horning's draft was accepted pretty much as is. M: Should we recommend that someone work on evaluation of systems? L: No, since no one in the group wants to.

Votes: M agrees to tabulate the votes. L and T have not yet voted.

We then continued down the catalog, and made it all the way through the *Tools* section in this meeting.

A1/B1: B1 is a subset of what we discussed last time about integrated sublanguages - see the minutes. After some fruitless discussion, it was agreed that we are happy with this, except that T would like B1 raised to A.

A3: Simple batch cross-reference facility is A, the rest is B. An incremental facility must be Fund. Further discussion revealed that the Fund. aspect is the need for a *single* funnel for changes to the system (Mesa pretty much has this now, but Lisp does not). Relation to the file system was discussed, and it was agreed that manual use of the file system should be outlawed. A consequence is that the PE must do recovery at least as well as the file system does. Of course, having a reliable file system underneath makes this much easier. A variety of techniques are possible, which we did not explore in detail.

A4: Often B6 is hidden under this - the two were discussed together. Definition: A4 is an efficiency issue: to get the right thing to happen without blindly recompiling and reloading everything. B6 is more fundamental, and was raised to priority A. It has two major aspects: history and parametrization.

Under history, we want to be able to tell exactly how a particular system or component was constructed, and what it depends on (e.g. microcode version, defs module or whatever). Furthermore, we want to be able to reconstruct a component automatically. This requires that every computation involved in its original construction must record *all* its inputs, and be prepared to repeat itself from this record. Since the inputs may be (references to) files, it is also necessary to have a naming scheme for files which is unique over the whole universe, and a guarantee that no file will ever be destroyed (unless the rule for reconstructing it is saved, together with all the required inputs).

Under parametrization, we want a systematic way of specifying the construction of a system which never existed before (e.g. it is for a new microcode version, or different implementations of the same interface are combined in a new way). We agreed that we don't aspire to solve this problem in the full generality required by IBM.

Replacing code in an existing system is in principle a special case of A4 - the general question is when a complete but expensive procedure (recompilation, reloading, etc) can be bypassed. It was noted that replacing code is in practice *not* just an efficiency issue, since getting the system back to the exact current state is not possible in general. The reason is that the current state depends on user program execution, and the user program cannot be counted on to follow the rules mentioned under history, which we impose on the system programs which make up the environment.

A6: Existing facilities were reviewed: Smalltalk and Mesa have a Spy, which works by sampling the PC, and Lisp has Breakdown. Proposed for Mesa by SDD are

> the Diamond Test Module's facility for counting frequency and time between any pair of breakpoints,
>
> a facility for writing an event in a log either by procedure calls, or as an action to be taken at a breakpoint, and
>
> a "transfer trap" mechanism which logs data at every control transfer, together with some standard ways of reducing this data to produce the same kind of information that a Spy produces.

It was agreed that something as good as Breakdown is good enough.

A7: The weakness of the facilities in all three current systems was noted: file state is not saved, nor is there any check that it hasn't changed on a restart. D: "protected environments" means the ability to install a new version of something in a system an still be able to revert to the old version very cheaply. If checkpoints are cheap, this would be B at most. Cheap checkpoints can be done in a paged system with copy-on-write techniques.

A8: D: this means a typescript file, the ability to feed back stuff from the typescript to the system, and the ability to have a handle on the values returned as well. M: All this is an attribute of the interactive interface. Undoing is of system-implemented actions (e.g. edits), and a way for the user to integrate with this, by supplying undo procedures for his procedures.

B3: H: Perhaps a lot of this is part of version control. L: DeSoto takes care of updating a "Bible" version from individual versions, and vice versa. H: That's what I meant. It was agreed that that is A. M: I want a "package librarian."

B4: Moved to A.

Distribution:
Deutsch
Horning
Lampson
Morris
Perlis
Satterthwaite
Teitelman

Inter-Office Memorandum

To	Distribution	Date	July 13, 1978
From	Jim Morris	Location	Palo Alto
Subject	Minutes of 7/12	Organization	CSL

XEROX

Filed on: <morris>pe-minutes-7-12.memo

D reported on progress with the write-up, lamented variation in depth.

D listed the various feature attributes: priority (A,B,C), votes/ranking, difficulty in various langauges, fundamental/add-on, and the enabling relationships.

D reviewed some sugesstion from Masinter (try Swinehart's non-premptive user interface) and Guibas (consider TEX, and the general idea of attaching attributes to text and other data structures). They will be put into an appropriate discussion section.

P asked whether the suggestion by Guibas was similar to the one that programs annotated by assertions be interpreted alternately by an evaluator or a verifier and pointed out that this idea had yet to bear fruit.

The outcome of the voting was discussed. H pointed out that the rankings according to median were roughly the same as according to the totals. This suggests that the effect of block voting was not significant.

H's categorization of fundamental/add-on was discussed. L and D revised many fundamental things to intermediate or add-on. The basic disagreement seemed to be whether "fundamental" meant the choice to include or exclude a feature was "important" or "non-reversible"

> L said that page memory management was add-on and object oriented swapping was not worth the trouble if there was enough memory.
>
> Cross reference capabilitues were raised to fundamental to the extent that the processing of programs had to be sufficiently centralized (unlike InterLisp) that there was a *funnel*; i.e. a process or program through which all the code passed on its way to becoming a system, or object worthy of being cross-referenced.

The Other section was dicussed with some revisions to priorities and difficulties. CSL control of the language was taken as given, although the extent to which we want to remain compatible with the various languages will be a hard issue, dependent upon the importance of the implementors and users of the langauges.

Packages, priority A, were discussed.

> The screen manager as an allocator of raster points was promoted to be part of the virtual machine, partly to avoid the current anarchy.
>
> The need for a package czar/librarian was cited.

D pointed out that the various images (text, line, bit) were but manifestions of some abstact objects that need to be specified and designed.

It was pointed out that it will be easier for a Mesa-based EPE to draw on Pilot for various packages; e.g. remote files.

Inter-Office Memorandum

To	PE Group	Date	July 17, 1978
From	Ed Satterthwaite	Location	Palo Alto
Subject	Minutes of 7/13 Meeting	Organization	SDD/SD

XEROX

Filed on: [Iris]<Satterthwaite>pe-minutes-7-13.memo

Warren was absent from this meeting; Alan Perlis attended.

What will happen next?

Peter will circulate a draft report for comments this week, and the report will be available the following week. We weren't sure what would happen after that (a CSL meeting was suggested).

We seemed to agree that each of the three languages were equally far from our goals ("about halfway there", by Jim H.'s estimate), and that the immediate roadblocks were clear in each case.

Jim M. asked about the increase in productivity that a EPE might provide. Butler estimated a fourfold increase (but part of this comes from the hardware in any case). Consequences for CSL might include fewer people per project, an increase in the number of systems attempted (because of smaller initial cost), and more ambitious projects at current project staffing levels.

We will meet again after Jim H. returns to discuss feedback received by then and to decide what to do next.

Discussion of Draft Report

Peter distributed a second draft of the proposed PE report. He commented on some gaps that he had noticed, mainly in explaining the rationale for various features. We agreed that Peter could use his discretion in expanding the discussions. We also noted that the specifications of the Lisp and Smalltalk systems we postulated in our evaluation needed clarification (especially re Pilot facilities).

Butler agreed to write a one page conclusion. We agreed that statements about productivity should be emphasized - where it will come from (packages, tools, language upgrades) and where it will go (more projects, more ambitious experiments, etc.). It also seemed important that the conclusions be stated in a way that will encourage discussion of the productivity claims rather than quibbling over the details of our choice and ranking of features.

We found that our agreement about the lessons we have learned from the existing systems is surprisingly high; most disagreement concerns the ranking of areas for further work. To support this claim, the report will include a tally of individual votes on the language features.

Peter agreed to write an introduction that will summarize the group's charter, how we proceeded, and how it all relates to the final report.

It is important that the report *not* be interpreted as a call for a new global plan.

Discussion of Mesa in SDD

Ed distributed a memo concerning the prospects for future development of Mesa by SDD. This provoked a somewhat extended and inconclusive discussion of how garbage collection might be added to (a restricted subset of) Mesa. Butler promised to resurrect and circulate an old memo on this subject.

We noted that the charter of SDD/SD Palo Alto includes producing a Mesa programming environment (albeit with somewhat differrent properties) and that SDD has committed ~6 people to the further development of Mesa (but pressures to produce a product could dilute this commitment). Lisp was seen as having goals appropriate for an EPE but a lower level of support, while Smalltalk seemed to have rather different goals.

Strawman Scenarios

We discussed several scenarios for developing a satisfactory PE. This included considering and rejecting a number of alternatives (developing Interlisp with some redesign, waiting for a Mesa PE from SDD, restarting from scratch). With respect to restarting from scratch, Peter agreed that much could be done with existing systems and that an evolutionary path to our long term goals seems possible.

The issue of a multilanguage EPE was raised repeatedly during this discussion. Peter and Butler were skeptical; they noted that an environment providing all our A features would support any existing style, while a two language system would reduce synergy and create artifical barriers ("one-language experts"). I believe we agreed that the only sensible course was to make a serious, single EPE effort that eventually would support all styles of programming. During the interim, all the other language systems must remain usable, but an attempt to extend all of them significantly would result in the commitment of subcritical resources to each.

Our evaluation of the existing systems seemed to be the following: there is not much to choose between Lisp and Mesa; the uncertainity in estimating the work required to make either satisfactory probably exceeds the differences between the two languages. Smalltalk is further from our goals but contains important ideas that should be captured. Smalltalk and Lisp are closer to one another than either is to Mesa, and Lisp might assimilate features from Smalltalk more easily.