UNIVERSITY OF CALIFORNIA, SAN DIEGO

The Cognitive Ecology of *Dynapad*, A Multiscale Workspace for Managing Personal Digital Collections

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Cognitive Science

by

Daniel S. Bauer

Committee in charge:

Professor Jim Hollan, Chair
Professor David Kirsh, Co-Chair
Professor Richard Belew
Professor William Griswold
Professor Edwin Hutchins

2006

The dissertation of Daniel S. Bauer is approved, and it is acceptable in quality and form for publication on microfilm:

[illegible]

[illegible]

[illegible]

[illegible]

Co-Chair

[illegible]

Chair

University of California, San Diego

2006

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Although no list of my appreciation for colleagues, friends, and family will ever be complete, I find it humbling and gratifying to place on the same page so many people important to this research and to me.

To begin, I gratefully acknowledge my funding sources: my host institution, the Cognitive Science Department of UCSD, for many forms of material support, and the National Science Foundation for grants #0113892 to Jim Hollan and #9873156 to Jim Hollan, Ed Hutchins, and David Kirsh.

I have enormous respect and gratitude for my dissertation committee. The intellectual godfather of this work is my advisor and committee chair, Jim Hollan. I am grateful to him for providing me more support, more opportunities, and more wise advice than I was ever able to harvest; for serving as a model of profound professional competence that I hope someday to imitate; for his patience and compassion through my phases of misdirection; and for believing in me and in this work even when I did not. My co-chair, David Kirsh, has been a friend and mentor from my first moment of contact with UCSD. He is to me both a figure of wit and warmth and a model for thinking deeply, working an idea downward past numerous false landings to hit a bedrock of clarity. I hope to achieve that even once in these pages. Ed Hutchins has stretched my mind to reach outside the head, giving me an appreciation for the material context of cognition and pushing me to new methodologies and ways of thinking. And I am grateful to Rik Belew and Bill Griswold for their enduring patience and receptivity and for pressing me on some difficult and important questions.

Dynapad has been a deeply collaborative project. All research is built on the efforts of others, but some software is particularly so; the process of design and development churns and mixes code from each contributor until each line is the work of many. It is therefore impossible for me to measure and itemize the enormous credit owed to my collaborators on Dynapad. However, the contributions of three colleagues in particular are pervasive. First: in countless instances, Ron Stanonik led the way into Dynapad's jungle, building the roads that I followed on. He single-handedly built the first Scheme interface to the original Pad++ core, vigorously implementing while I fretted about how to begin. Ron has been a programming juggernaut: patiently, methodically, and cheer-fully crushing every technical problem in his path. Second: wherever there is elegance in Dynapad's hidden entrails, Dr. Ron Hightower deserves much credit. Dr. Ron introduced me to design patterns and honed my engineering aesthetic. His vigilance and disciplined craftsman's eye often saved my from my own worst programming instincts, and my episodes of close collaboration with Dr. Ron have been among the most fertile of my professional life. And finally, I must again credit Jim Hollan, this time in his technical roles as producer, director, and master architect of Dynapad. For all of my exploration on Dynapad's established themes, Jim's vision inspired those themes. In name and in scope and in many small details, Dynapad is Jim's creation, built upon years of his work with Ben Bederson and colleagues.

Additional contributions to Dynapad's pool of code resources were made by Shaun Haber, Etienne Pelaprat, Ben Shapiro, and Dana Dahlstrom. Especially significant was Dana's work; he implemented critical components of the DiamondTouch table interface and the first version of generalized brushing. And more generally, I've appreciated Dana as a sounding-board and assistant in the later stages of development. The DiamondTouch table itself was generously donated by Mitsubishi Electric Research Labs. Pierre Fastrez and Dev Yamakawa devised and conducted the PhotoPad observations, paving the way for both the protocol and analysis of my own later observations. Pierre especially has been an energetic, imaginative, and amiable collaborator.

I am uniquely indebted to my two colleagues and research subjects, **A** and **B**, for their generosity, patience, creativity, and fortitude. They graciously donated their own bytes and many hours and turned the rickety props into a compelling show of cognition.

More broadly, I have benefited in countless ways from the intellect and enthusiasm of innumerable members of the Cognitive Science department and the DCOG-HCI lab, including Morana Alac, Amaya Becvar, Monal Chokshi, Deborah Forster, Ankur Jalota, Christine Johnson, Marta Kutas, Jim Levin, Saeko Nomura, Thomas Rebotier, Beate Schwichtenberg, and Bob Williams. Special thanks to Ron Stanonik and Mark Wallen for many years of patient expertise, technical support, and generous disk quotas, and to Aaron Cicourel, for quietly watching out for me all that time. I am also indebted to a series of brave graduate coordinators: Gris Arellano-Ramirez, Becky Burrola, and Beverley Walton, whose collective administrative competence often saved me from my gravitation to disorder.

My experience of graduate school, while often deeply rewarding, has also been a marathon with an elusive finish, and my well-being and morale are intact only from the enduring support of dear friends. My network of support over the years extends far beyond my enumeration here, but I'm compelled to highlight a few names. I am deeply grateful to: Mike Hayward, a great friend, lab-mate, and hipster extraordinaire, my role model for surviving graduate school while still living well; Scott Herscher and Carrie Joyce for cherished evenings of music, mirth, and Henry the porcine aviator; Jacquie Lowell and the agents of *Mission:Improvible* for nurturing joyful stoopidness and helping me laugh despite myself; Esther Pascual, for warming me with relentless optimism and a cosy hat; Kim Sweeney Wolanyk, a treasured friend, colleague, and commiseratrix through many seasons, for giving me many carrots and the occasional stick, and for Sioux, who trained me in the Joy of Dog and convinced my face it needed licking. I owe a special debt to my great-uncle Billy, who generously and clairvoyantly funded my first computer, twenty years early.

And finally, as always, I return in the end to my precious family: my parents, Cynthia, Ward, and now Katherine, my sister Laura and brother-in-law Scott, and the memory of my brother Doug, who might have done this first. From them I've been granted the extraordinary privilege of feeling loved no matter what, of being nurtured in a home of limitless freedom, opportunity, and encouragement to become myself. Thank you for the many years of loving support, cheer, and patience through my waves of surliness and dismay and long absences while pursuing this elusive sciencey-computery thing.

VITA

1993	B.S., Symbolic Systems, Stanford University
2000	M.S., Cognitive Science, University of California, San Diego
2006	Ph.D., Cognitive Science, University of California, San Diego

PUBLICATIONS

Daniel Bauer. Personal information geographies. In *CHI '02 extended abstracts on Human factors in computing systems (CHI'02 Doctoral Consortium)*, pages 538–539. ACM Press, 2002.

Daniel Bauer. A multiscale workspace for managing and exploring personal digital libraries. In *Proceedings of the 16th ACM Symposium on User Interface Software and Technology (UIST'03 Doctoral Symposium)*, November 2003.

Daniel Bauer and James D. Hollan. IRYS: A Visualization Tool for Temporal Analysis of Multimodal Interaction. In *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI'03)*, pages 285–288. November 2003.

Dan Bauer, Pierre Fastrez, and Jim Hollan. Computationally-enriched 'piles' for managing digital photo collections. In *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing (VLHCC'04)*, pages 193–195. IEEE Computer Society, October 2004.

Daniel Bauer, Pierre Fastrez, and Jim Hollan. Spatial tools for managing personal information collections. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS'05)*, page 104.2. IEEE Computer Society, 2005.

ABSTRACT OF THE DISSERTATION

The Cognitive Ecology of *Dynapad*, A Multiscale Workspace for Managing Personal Digital Collections

by

Daniel S. Bauer
Doctor of Philosophy in Cognitive Science

University of California San Diego, 2006

Professor Jim Hollan, Chair
Professor David Kirsh, Co-Chair

Dynapad is a prototype software application designed to support users in organizing and exploring personal collections of digital information. It provides a continuously zoomable workspace, essentially an infinite desktop, on which users can arrange digital photos and document "portraits", visual summaries of collected PDF documents.

To support users in engaging with their collections, Dynapad offers a unique combination of affordances. These include: 1) "brushing" media objects to reveal and highlight others related by various metadata; 2) interactively revisiting any episode in the history of one's workspace; and 3) employing portable "region-tools" which are computationally enriched with various forms of localized automation. An important special case of these region-tools is a digital "pile" which emulates — and augments — the affordances of paper which make "piling" a ubiquitous complement to "filing" as an organizational strategy.

The resulting environment is flexible enough that users are left considerable freedom and responsibility to invent organizational strategies and to structure their own activity. The tactics they adopt are emergent and exaptive behaviors shaped by small details of Dynapad's design, the affordances and cues which together constitute the environment's "cost structure" for interaction.

My research explores that cognitive ecology through detailed exploratory observation of two Dynapad users working with collections of their own digital documents. To analyze those observations, I trace a network of influences from design details to the behaviors they shape. My representation of that network identifies and dissociates affordances that participate differently in the cost structure, particularly in their contributions to piling. It is not meant to provide a predictive model of users' behavior; instead, it offers a theoretical framework for interpreting and synthesizing my and others' observations. Such a description is a necessary component of the understanding required for effective redesign of Dynapad and the design and development of other interactive systems.

1

Dynapad and the Ecology of Design

Auguste Rodin's famous statue *The Thinker*, inert and introspective, is complicit in a lie; sitting naked is no way to think. The vast majority of "thinking", of cognitive activity, happens not in static rumination but in close interaction with the physical world, engaged with tools in malleable spaces. Our behavior is largely a reaction to our environment; we adopt strategies that the environment makes easy and avoid those it makes difficult. We rearrange the world to make us smarter. This is the core principle of design: shaping the world to enhance ourselves.

Software engineering has a unique position among design fields because, for the first time, we are gods: we have control over the design of the most basic physics of virtual worlds. This leaves us all the more opportunity and responsibility to design not simply an artifact or environment but an *interactive experience.*

This is the story of the design of a virtual environment, Dynapad, and the experience it creates.

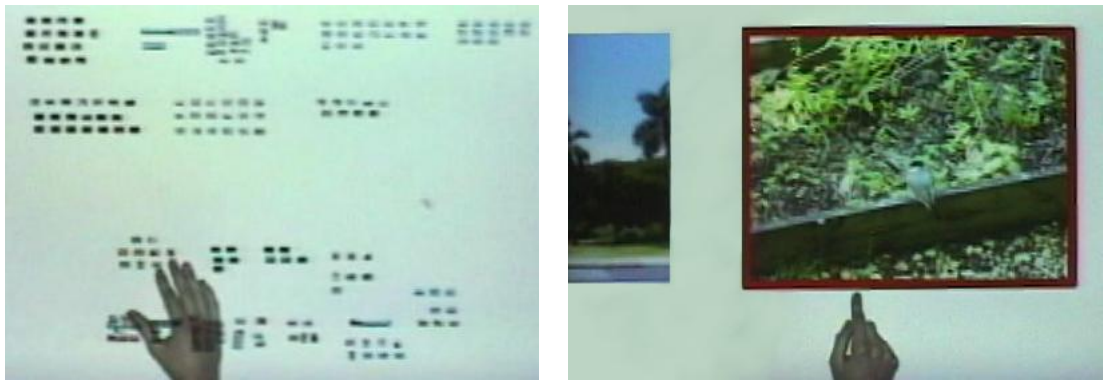
1.1 The Problem of Managing Stuff

Since bytes are cheap, everyone faces a deluge of digital information. It's bad enough that public resources (e.g. libraries, the Internet) become overwhelming, but at least those are somebody else's problem to manage. For most of us, a bigger challenge is to manage *personal* collections of digital information.

One aspect of the problem is *retrieval*, locating particular items on demand. In addition to a long lineage of retrieval tools on personal computers, there are now web services (e.g. gmail, Google Desktop, Picasa) which let individuals “outsource” the effort of archiving, indexing, and retrieving personal data.

Getting the Big Picture. But the challenge of collection management isn’t just one of retrieval but of *understanding the whole* of what we have. The process of organizing a collection manually offers two benefits: the eventual organization, but also the process itself. By interacting with a body of materials, we make discoveries, grow familiar, develop expertise, and learn to see things in new ways.

Dynapad is a virtual environment supporting not only the organization but the exploration of personal digital collections, to deepen our understanding of our own materials.



(a) Zoomed out (b) Zoomed in

Figure 1.1: Snapshots of interaction with a photo collection in Dynapad. Here the workspace is projected onto a tabletop, and the user is gesturing over it.

1.1.1 A Very Brief Introduction to Dynapad

The chapters ahead will discuss both the motivation and features of Dynapad in great detail. But this chapter is concerned more broadly with the *process of design* and will introduce Dynapad only enough to establish its role as a designed artifact in that process.

For the moment, we can think of Dynapad as a software application, an integrated suite of tools with a user interface. Dynapad creates a virtual spatial environment, a *workspace*, much like the "desktop" of many windowing systems. The workspace holds a *collection* of digital information such as photos or documents. The user can browse and rearrange these items, supported by various organizational tools, to explore and manage the collection.

The User's Artifact: the Workspace. Over a period of interaction, the user rearranges and annotates the contents of the workspace. For example, a typical strategy is to organize the collection into various "piles",[1] each containing a particular category of material. The organization in Figure 1.1(a) includes many such groupings, which may be arranged implicitly, labeled explicitly, or annotated in other ways. This process gradually structures the workspace as a *representation* of various relationships and themes in the collection. We can think of this representation, the workspace itself, as a *digital artifact* the user develops over time, much like a document produced by a word-processing program.

The Designer's Artifact: the Tool. The Dynapad application itself has the same role as the word processor: it is the software tool with which a user constructs their artifact. Like all software, Dynapad is implemented as a body of code (the program), which is itself an artifact crafted by the programmer.

In some sense, both the programmer and user are engaged in processes of design, the gradual refinement of their respective artifacts. But these two processes are very different and require different terminology. Throughout these chapters, "designer" will mean the programmer, "design" will mean the program or the process behind it, and "activity" or "interaction" will mean the user's engagement with their own artifact, their virtual workspace.

So far, we've characterized Dynapad as a particular application, but more precisely it is an *infrastructure* or *toolbox*, a set of programming resources and design ele-

[1]Here the word "pile" is intended in a loose sense. In Dynapad, "piled" items are typically grouped together but not stacked. Such differences are discussed exhaustively in Chapter 2.

ments which can be combined in different ways to construct different applications. We can think of each particular application, each configuration of features, as a *design instance* or *design variant.* So Dynapad is really a *family of designs*, related to each other by shared concerns, resources, and constraints. The term "family" is meant in a very literal sense: not just a set but a *lineage*, whose members descend from and inform each another through an ongoing progress of refinement.

Two Cycles of Change. The gist of this scenario is that there are two participant roles, each developing an artifact at a separate time scale. A *designer* shapes Dynapad's code and slowly evolves a family of design variants. With a particular variant, a *user* develops a personal artifact, their workspace. These two processes are neither independent nor strictly nested. In Dynapad's case, as we will discuss later, the design evolved in parallel and in response to the users' ongoing participation. But for the moment, we may regard the development of a user's artifact as nested within one iteration of the longer-term cycle of design, as Figure 1.2 depicts.

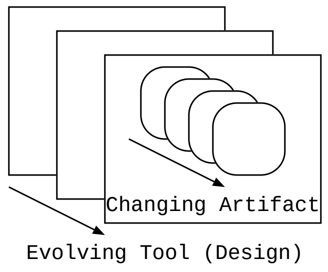


Figure 1.2: Two time scales of change: a user interactively develops her artifact within one instance of the designer's more slowly evolving tool.

1.2 A Framework for Design

The rest of this chapter will refine this basic picture into a more sophisticated model of the interplay between these two scales of activity. This model is not intended to be a comprehensive explanation of the complex interaction of people, activities, and materials involved in design. Nor does it mean to challenge or revise theories of any of the constituent phenomena it attempts to synthesize, which have been explored in greater detail in others' work (cited ahead where appropriate). Instead, this model's purpose is to frame the particular story of Dynapad's design, to act as a conceptual scaffold on which to hang the elements of that story, detailed in the chapters ahead, as they are fit together.

The central thread of this chapter is the gradual refinement of two related questions:

Q1: When is a design good?

Q2: How do we make it that way?

These may sound redundant, but later we'll see their important difference.

1.2.1 Generating and Evaluating Designs

In his seminal work "Sciences of the Artificial" [68], Herbert Simon reaffirms Design as a coherent and principled intellectual practice. Design, he argues, has been marginalized as "soft" and "cookbooky" in academic communities which favor "pure" theoretical and analytical styles of inquiry. But the great variety of professions, both academic and "applied", which practice design tacitly share a theoretical foundation which Simon attempts to articulate.

A Design Example: Edison's Lightbulb. Simon's formulation, elaborated ahead, will be clearer with an example. For this purpose, let's draw on an actual historical design episode: the development of the incandescent light bulb. For simplicity, let's consider just one dimension of the bulb's design, the choice of filament, pretending that other aspects (e.g. the evacuated bulb, the switch, and the electricity supply) are fixed. Even within these limits, the design space is huge: Edison and others tried thousands of potential filaments, of varying materials and diameters, in an effort to achieve both brightness and longevity. In 1879 Edison's best candidate, a carbonized bamboo fiber, burned for 40 hours.

Simon's *Means*, *Ends*, and *Laws*

With this example in mind, let's return to Simon's formalization of Design. Simon begins from the domain of Optimization Methods, which is concerned with finding the "best" solutions to quantitative engineering problems. Simon's paradigm, in a nutshell, is that a design is a *means* to an *end*, subject to certain *laws*. His model's components are reflected in these three terms: means, ends, and laws.

Means (Design Instances). A design's *means* are simply the properties which constitute it: what it does and how. Simon calls these *command variables* (i.e. variables which the designer commands). They might be formalized as *attributes* with *values*. A set of command variables or attributes defines a space of possible designs, and a particular set of values distinguishes one design instance from others. Searching for a design means deciding on the values of these variables.

In our lightbulb example, the command variables of the filament choice are material (e.g. carbonized bamboo) and shape (i.e. length and diameter). These attributes define the search space, the scope of the design. A particular set of values (e.g. Material=*bamboo*, Length=*5mm*, Diameter=*0.5mm*) instantiate one design, one "means" to the goal of incandescent lighting.

A particular design, a set of these values, defines an *inner environment*: the space of possible actions available to someone using it. A trivially simple case can be seen with our light bulb example. A light bulb is not what we would think of as a typical tool, since its function does not demand the user's continual engagement. But it still defines a very simple inner environment: the user chooses when to invoke its function by turning it on and off. In the simplest case, this inner environment has only two states (on and off) and the two corresponding actions.

As a complementary example, consider a modern three-way bulb (and accompanying switch), which offers a four-state inner environment (three brightness levels plus "off"). Or if we extend the boundary of the design to include multiple bulbs (possibly of different intensities) and multiple fixtures, then the user's inner environment includes decisions about what bulb to use where, as well as when to activate each.

Laws (Extrinsic Constraints). Simon's *laws* are constraints on the design imposed by the external world, the *outer environment*. These are assumed to be outside the control of the designer (and users); otherwise they can be manipulated as part of the design.

In the lightbulb example, the laws include the immutable physical properties of materials which dictate both the attainable diameters and lengths for certain materials and the trade-offs between intensity and longevity for each configuration. And if we restrict the design problem to include only the filament, then potential variables such as electrical voltage and bulb vacuum quality can also be considered fixed "laws" in that scenario. (In practice, of course, the boundaries of a design are often negotiated to include revision of such assumptions.)

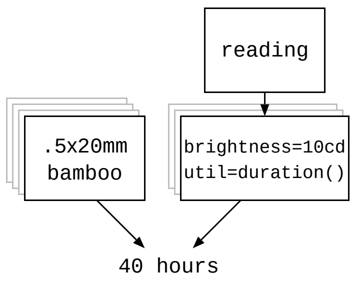
Ends (Goals and Evaluative Measures). A design's *ends* are its goals, what the designer intends to accomplish. We can *operationalize* these goals as an evaluation scheme. In Simon's model, the evaluation scheme has two facets: constraints and utility. The *constraints* are fixed criteria that should be satisfied by any design. The *utility* or *goodness* represents *preferences* beyond these minimum requirements, by which one design may be judged better or worse than another. To make such comparisons, the utility must be quantified by a *utility function*, which gives a utility value for each possible design. An optimal design is one which meets all constraints and has the highest possible utility.

The reason for this formulation is that it guarantees a reliable *decision rule* for comparing designs. A quantitative utility function creates a transitive relation between possibilities, so that comparing any two designs can eliminate the weaker one from consideration, allowing steady progress toward an optimal design.

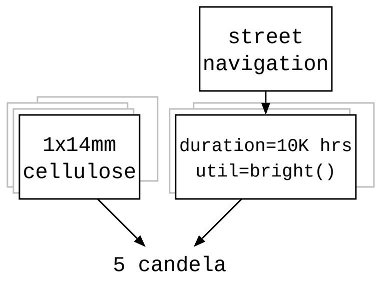
Consider again the light bulb. It general purpose, clearly, is to illuminate, but this has two competing facets: brightness and longevity (let's ignore a third obvious dimension of affordability). If one filament is bright but burns out quickly, and another is dim but lasts longer, which do we choose? It depends on what we choose as constraints and utility, and that depends on exactly how the bulb will be used.

Suppose the user needs lighting for a short-term but highly visual activity (e.g. reading) which requires a minimum level of illumination. We might then designate that minimum brightness as a fixed constraint and the bulb's longevity as a utility function (i.e. the longer the better). Figure 1.3(a) illustrates this situation.

Now suppose instead that the bulb is to be used to light streets for nighttime navigation (Figure 1.3(b)). In this case, we might demand that such hard-to-access bulbs



(a) For nighttime reading, we might demand a minimum brightness and choose duration as a utility function to compare filament choices (e.g. bamboo).



(b) For street navigation, we might demand instead a minimum duration and choose brightness to compare filaments. In this case, a different candidate (e.g. cellulose) might yield higher utility.

Figure 1.3: Two hypothetical choices of filament and evaluation scheme. The evaluation scheme must reflect the user's goals. Each combination yields a utility value.

last a minimum duration, and then compare their relative utility by brightness.

Note that any particular design (i.e. filament choice) may have a different relative utility depending on which evaluation scheme we choose. That is, the utility of a design is the product of two potentially independent choices: the design itself, and the standard of evaluation. We might represent this schematically as in Figure 1.4.

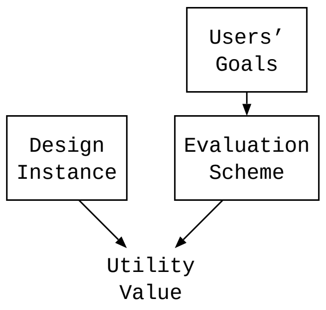


Figure 1.4: The utility value of a particular design instance depends on the evaluation scheme (constraints and utility function), which should support the users' goals.

Of course, in practice these choices are not made independently; a thoughtful designer will select designs *anticipating* their value by established utility schemes. And conversely, designers face a temptation to choose evaluation standards which reflect well on the designs they favor.

And we must remember that neither choice is fully in the designer's hands; both are constrained by laws outside the design space (see Figure 1.5). The incandescent filament's design space is constrained by material properties that dictate whether, for example, bamboo fibers may be thinner and longer than, say, extruded cellulose (another of Edison's candidates). Such laws also constrain the evaluation standards — for example, by disallowing criteria which cannot be physically achieved (e.g. bright and long-lasting and cheap all at once).

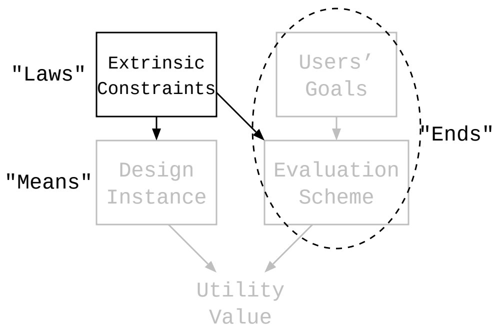


Figure 1.5: An abstraction of the dependencies in Simon's basic model.

1.2.2 Waiving Optimality

Figure 1.5 shows the basic dependencies of Simon's model. All dependencies flow downward: a design's utility value is a product of both the design and the goals, but the goals do not change as a result of the design. Later we will challenge this assumption, thereby changing the overall dynamics of the model.

Satisficing Criteria. Simon recognizes that this initial formulation is simplistic in many respects, and he immediately relaxes the goal of finding *optimal* solutions to finding merely *satisfactory* or *sufficient* ones. Combining these words, he coins the term "satisfice": a design is judged not to be optimal but to *satisfice*, to be *good enough.* (Extending Simon's neologism, we might also describe the design as "satisficient", exhibiting "satisficiency".) The simplest form of such a decision rule still relies on an ordinal utility function: a design *satisfices* if it meets all constraints and has at least a certain minimum utility. But Simon's reformulation obviates the need for a utility function. Instead, the goals of a design are abstracted into *satisficing criteria.* The decision rule or *evaluation* of a particular design is simply whether it meets whatever satisficing criteria the designer adopts. Although these criteria need not be quantitative, they must still be *operationalized*; that is, there must be some explicit procedure for making the decision.

Steering Mechanism. Even with a decision rule for evaluating particular designs, these designs must come from somewhere. Almost always a designer doesn't postulate designs blindly but with educated *heuristics* about what will prove satisficient. Simon abstracts this component as a *steering mechanism*, a process of reflecting on the relative satisficiency of various designs to guide the search for better ones.

Note that these two design considerations correspond to the two questions posed earlier. They refine our inquiry as follows:

Q1: When is a design good?

$\longrightarrow$ Assuming a user's goals are fixed, does the design satisfice?

Q2: How do we make it that way?

$\longrightarrow$ What mechanism will steer redesign?

The Design Cycle

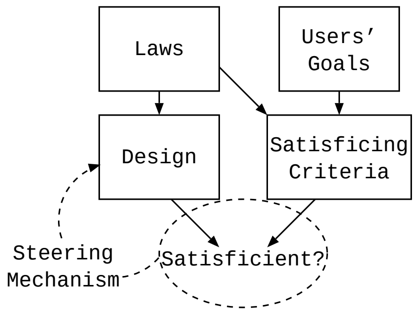


Figure 1.6: Simon's refined model; a steering mechanism guides new designs by considering how a design *satisfices.* This is the basic *design cycle.*

Upgrading the earlier graph to reflect these additions yields a picture like that of Figure 1.6. The flow of influence in this structure is not strictly forward but cyclic: each design leads to another, mediated by an understanding of how each satisfies the user's goals. This picture merely makes explicit the iterative process behind all designs.

But this representation also highlights a critical assumption behind Simon's formulation: that the goals which motivate the entire process are outside the loop, remaining fixed as the design evolves.

How can we modify Simon's formulation to describe a design process with a moving target?

"Generate and Test" in HCI. Borrowing from various empirical sciences, much work in HCI includes conscientious efforts to evaluate and compare designs quantitatively. A typical pattern of such research is "Generate and Test": first building several design variants, and then comparing them according to a predetermined "goodness" measure (e.g. speed. effectiveness, effort, user satisfaction). Ironically, this emulates the crudest form of Simon's model, with a prescribed utility function rather than the more general satisficing criteria.

1.2.3 The User's Activity

Simon's model is intentionally very broad, applying to design of all types (toasters, algorithms, etc). But Dynapad belongs to a particularly important class of designs: those which support prolonged *interaction.* And for these, Simon's model makes no explicit reference to what is the central consideration of such designs: the *activity* through which they are put to use. Our next step is to expand this model to include some role for the user's activity.

Affordances

In constructing a theoretical framework for understanding activity, let's begin with the notion of an *affordance*, introduced by Gibson [29]. An affordance is an "opportunity for action" [43]; an object *affords* certain actions to certain participants. For example, a closed drawer affords pulling open — but only to someone who is able to grip it in the right way. Once open, the drawer affords pushing closed — but again, only to one who can do so (e.g. who is strong enough and has a free hand or perhaps manages to use a knee or elbow). These examples illustrate that an affordance is not a property intrinsic to an object but a three-way relation between the object, an action, and an actor.

The key point is that the affordances around us *shape our activity* by making some actions easy and others difficult or impossible. So, at least in principle, we can design activity by designing the appropriate affordances into the environment.

Additionally, to the extent that we can perceive them, some affordances not only permit but *suggest* actions. For example, the clasp of a drawer or cupboard door makes gripping and pulling a very visible and salient option. But some drawers and cupboard doors have no handle; they are opened by first pushing in to release a hidden catch. They *afford* opening by pushing, but they don't *suggest* it; we must rely on prior experience or exploration to discover that.

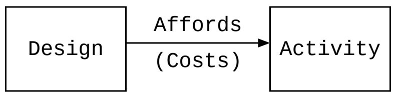


Figure 1.7: Designs afford certain activities by inducing various costs of action.

This basic principle of an affordance, as described here, can be generalized in three ways.

Costs are Continuous and Qualitative. First, affordances need not be binary (possible or impossible) but may be continuous: an object affords or cues an action with some degree of effort. Every action has a cost (possibly infinite). It will be more apt to discuss the *costs* of activity rather than merely the possibilities. And eventually we should consider not merely the magnitude but the *kind* of costs. The cost of action is not merely a quantitative but a qualitative measure.

Actions may be Complex. Second, especially in the digital world, objects and actions may be non-physical. In a virtual environment, our repertoire of action is determined not by our physical bodies but by the input "vocabulary" of a particular interface. Therefore a virtual "button" affords "pressing" by the relatively small effort of moving and clicking a physical mouse. By extension, we may think of such affordances, especially low-cost ones, as primary actions with the potential for further affordances: a window affords "closing" by pushing its close button.

But this risks becoming a slippery slope where the definition of our primary repertoire grows to include every potential complex interaction. For example, it seems inapt to claim that a pantry door affords making dinner. But this is one important reason to define an affordance as a *graded* rather than an all-or-none relation. The more complex an activity is and the more elements it involves, the weaker the contributions of particular physical properties become. The pantry door's properties contribute to dinner-making, but they impact only a tiny portion of the total cost of the activity. If we are to consider this as an affordance, we must qualify it as a very weak one.

And yet, a thoughtful design of a dinner-making environment must consider such details in how the door participates in that activity. Ultimately, the designer's goal — and the goal of this thesis — is to bridge that gap between physical details and the complex interactions they indirectly support or inhibit.

Elements may be Collective. This brings us to the third generalization of affordances: they must be considered in their context. Objects and actions are rarely isolated but parts of ensembles: objects constitute *environments* and actions constitute *activity.*

Speaking loosely, we might then say that an environment affords various activities. Likewise, considering a design as an "inner environment," a design affords various activities. But the notion of a simple cost, which a single object incurs for a single action, must be extended when we interconnect a system of objects and actions, mapping a environment to an activity. The relation between two sets must itself be multiplex. I'll call this relation the environment's *cost structure.* Our earlier lemma then becomes: *the degree to which a design affords certain activities depends on its cost structure.*

Cost Structure

In one of the earliest uses of the term, Card, Robertson, and Mackinlay [14] consider the cost structure of information in a typical office. Information is stored in various locations and formats: papers on the desk (immediate storage) are easy to access, files in cabinets (secondary storage) are more difficult, and library archives (tertiary storage) require even more effort. The physical layout of the space imposes differential costs for accessing information. The individual affordances of the storage areas constitute (or at least determine) that environment's *cost structure* for seeking information. The same environment has a different cost structure for other activities (e.g. dusting) and for other actors (e.g. someone with impaired mobility).

Activity Landscapes. We can think of the office as a *landscape* in which different areas incur different costs. Simon himself [68] appeals to a similar metaphor: an ant exploring a beach will trace a complex path, not because of any complexity within the ant, but as a reflection of the complexity of the beach. The ant's physical landscape shapes its activity by inducing a corresponding virtual "landscape" of costs and rewards.

We can generalize this principle by adopting a more abstract definition of an environment. As Simon suggests, the "inner environment" imposed by a design is most often not a physical space but a *problem space* or *activity space*. That is, the user solves a "problem" by changing the state of her inner environment or "artifact"[2] to some "goal" state. The possible states are "locations" in this abstract space, and the possible actions are the "moves". The participant tries to "travel" to the goal state while, like Simon's ant, continually negotiating the local terrain. The environment's cost structure determines that terrain: the difficulty (and *consequences*) of various moves from various states or conditions. This cost structure has therefore been called an *activity landscape* or *affordance landscape* [43].

Cue Structure

Earlier we considered two aspects of an affordance: what it allows and what it *suggests*. We've considered primarily the former, such that a cost structure reflects the costs of *acting* in various ways. But to reflect further the variability in the ways an environment *suggests* actions, we need the complementary notion of a *cue structure* [43]. An environment's cue structure comprises various costs of *perception* and *attention*:

- the natural visibility and relative salience of affordances;
- deliberate manipulation of that salience, by either the designer or user, as a way of encoding heuristics and constraints on activity. Kirsh [42] offers many examples,

[2]Here "artifact" is meant in the broadest possible sense: it may be an actual mutable object or representation (e.g. a paper document or Dynapad workspace), the configuration of one's physical environment (e.g. opening doors, moving objects) or oneself (e.g. body position), or even a non-physical representation or conceptualization (e.g. mental computation).

which include:

  - highlighting affordances or choices which are known to be apt in a particular state;
  - hiding affordances or choices which are known to be inapt;
  - making visible the relative value or cost of one's choices;

- feedback on the progress or status of an activity, which Kirsh calls "task regulating attributes" or simply "indicators" [43];
- representations to simply perception or computation [42].

These aspects of cue structure are closely interrelated, but their relationships are not important here. I use the term "cue structure" very broadly to include any aspect of an environment which biases activity that is difficult to reconcile with the (also loose) meaning of "cost structure". If cost structure includes, for example, the direct impact of an environment during the user's actions, then the cue structure should encompass the environment's role in the "whitespace" between actions.

The key point is this: cost structure and cue structure together introduce a landscape which shapes the activity of participants. They are a product of both the design and the physical laws and constraints imposed by the outer environment. But within these constraints, cost and cue structure can be manipulated *indirectly* by altering the design. Cost and cue structure constitute the medium through which a designer participates in shaping a user's activity.

In the discussion ahead, cost structure and cue structure often play the same role in the design ecology and need not be dissociated. In such cases I'll often refer to them collectively as "cost structure".

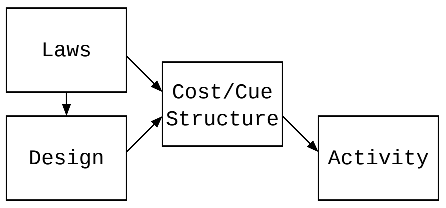


Figure 1.8: Designs impose a cost structure and cue structure which shape activity.

1.2.4 Negotiating Goals in a Reflective Cycle

Of course, activity is not dictated completely by the affordance landscape; a user does not mindlessly follow a path of least resistance, but generally works toward certain goals or intentions. In refining Simon's model, we've temporarily set aside that key component, the user's *goals*, which define the "ends" a design means to serve. Now let's reinstate that consideration.

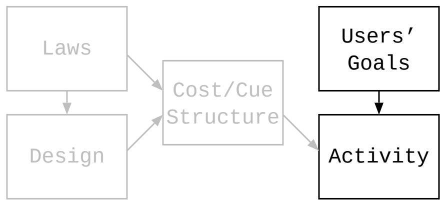


Figure 1.9: Activity is a product of both the user's goals and the environment's cost structure.

Earlier we noted that an implicit assumption of Simon's formulation is that the goals which give a design its purpose are fixed. Now it's time to challenge that assumption.

Reflective Feedback

Consider for the moment a particular class of designs, *tools*, which support a user in developing an artifact, either physical or virtual. For example: a woodworking tool shapes furniture, a word processor generates documents, and Dynapad yields a spatial organization of documents. As suggested earlier (Figure 1.2), in these cases the user is acting as a designer in an "inner" activity embedded in an environment which is itself the product of an "outer" design activity. We've seen that the outer activity has a cyclic nature: the designer reacts to and reflects on the satisficiency of his design (the inner environment). So we should expect that the inner activity has an analogous cycle.

Such a cycle has been examined in many others' work, especially that of Donald Schön [63]. Schön characterizes design activity as a "reflective conversation with materials" in which the designer continually reacts to new discoveries in her developing artifact. Schön distinguishes two time courses of such feedback. *Reflection*-**on**-*action* is a designer's retroactive discovery and consideration of a design's implications. *Reflection-*

in-*action* is a real-time awareness of those implications, where the designer's instinctive "flow" changes to conscious improvisation. This distinction in timing is not important for our purpose here, which is merely to recognize the presence of "representational talk-back" [73] from the artifact produced by activity to the *intentions* which guide that activity. That is, the participant's goals evolve in continuous negotiation with that activity and its artifacts.

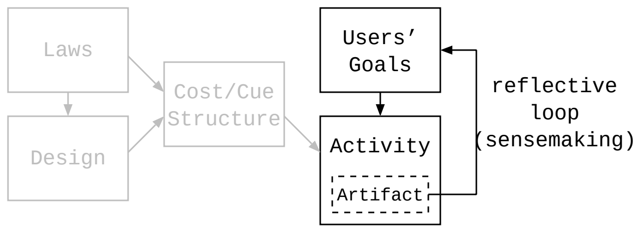


Figure 1.10: The user's reflective loop. Goals, activity, and mediating artifacts co-evolve.

Sensemaking. This basic dynamic applies not just to activities that are explicitly "design" but potentially to any activity. As a participant interacts with the inner environment (the "problem space"), continually articulating her understanding through action and re-interpreting the result, she gradually *refines* that understanding. This process has been called "sensemaking" [58]. Although defined very broadly, the term is meant to imply more than merely "learning". "Sensemaking" emphasizes that the conceptual framework required to assimilate information is itself under continuous revision. In effect, sensemakers *discover what it is they're trying to learn or accomplish.* Therefore their *goals*, like a designer's, evolve in a reflective cycle.

Cost Structure and Goals

We've seen that a design's cost structure influences a user's activity. Some of that influence we can ascribe more precisely to an impact on the user's goals. Specifically, cue structure makes opportunities differentially visible, and cost structure makes them differentially appealing. The user will adopt salient goals and easy strategies and avoid difficult ones.

For our purpose here, it doesn't matter whether the cost structure impacts more directly goals or activity. The model of Figure 1.11 is deliberately simplistic,

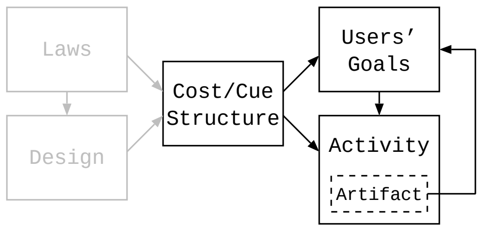


Figure 1.11: The inner environment's cost structure mediates the reflective loop.

with components which are inadequately dissociated. But it illustrates the essential dynamics: the user's behavior is a dynamic process which is largely self-driven, but the environment's cost/cue structure has the potential to *perturb* or *mediate* that loop.

The Designer's Reflective Cycle. We've identified reflective feedback at two levels: the user's loop is within the "inner environment", and the designer's loop is iterated redesign *of* that environment, the same basic relationship shown in Figure 1.2.

This suggests another refinement of the designer's "goodness" measure:

Q1: When is a design good?

$\longrightarrow$ Assuming a user's goals are fixed, does the design satisfice?

$\longrightarrow$ Does the *cycle of goals and activity the design imposes* satisfice?

The designer's only means of intervention is indirect: changing the design so that its cost structure induces a satisficient cycle of behavior.

1.2.5 The Context of Activity

We've seen that a user's goals are a mutable product of activity. But surely they're not *completely* mutable; if they change drastically, at some point the activity they induce will no longer be appropriate. But appropriate to what? What motivates those goals in the first place?

The answer is that the user's activity is always situated in the context of some "outer" activity taking place in the design's "outer environment". I'll refer to this outer activity as the *practice* for that design.

As an example, consider Dynapad. Dynapad is a design, an environment with a cost structure. The resulting *activity* includes whatever users do in that environment,

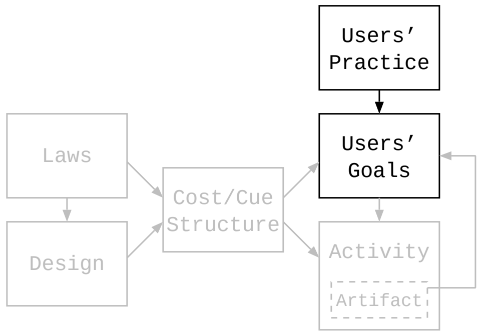


Figure 1.12: The user's practice motivates the goals

while interacting with Dynapad. The surrounding *practice* includes all of the other habits and demands in the user's life for which Dynapad *matters*, the pattern of when, how, and why Dynapad is used. We can imagine many examples:

- Using Dynapad as a "shoebox" for digital photographs: throwing them in periodically, with little invested effort, and once a year browsing through them to find selections for a homemade holiday newsletter.
- Using Dynapad extensively and long-term for multiple media types, as a visual interface to one's file system.
- Using Dynapad intermittently throughout an intensive summer workshop to manage its copious reading material, then revisiting it years later while looking for a particular article.
- Using Dynapad heavily for a week to organize an accumulated collection of papers into an annotated bibliography, then adding a paper occasionally afterward.

The satisficiency of a user's evolving activity must be evaluated with regard to the practice which motivates it. Loosely stated, the activity must be compatible with its practice. Borrowing a term from experimental research, we might call this the *external validity* of the activity (or the design which invokes it).

Negotiated Usage: The Evolution of Practice

Any of the example practices above will prime a user with certain goals during usage. Although those goals may change in the reflective cycle, as the user navigates the design's cost structure and reacts to her developing artifact, the goals will retain some loyalty to the practice which motivates them.

And yet the practice itself is subject to change. As a user continually reinvents her inner activity (e.g. trying new organizational strategies in Dynapad), she also discovers what else the tool (Dynapad) can be used for — that is, she renegotiates her usage practice. Thus there is a third cycle in the design ecology: *negotiated usage.*

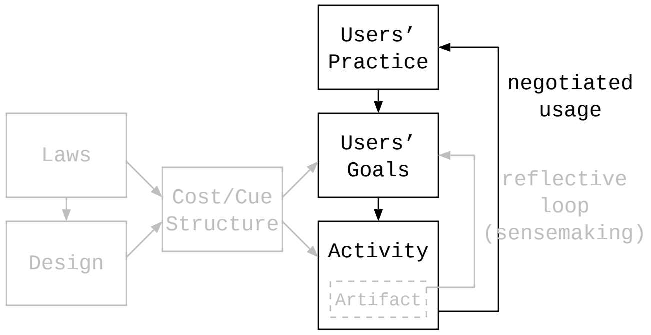


Figure 1.13: The cycle of negotiated usage

Interacting Practices. As an activity itself, the practice participates in its own local ecology. Whatever that environment is, it has a cost structure of its own, which may itself be the product of design. Figure 1.14 shows multiple levels of activity, where each serves as a context for the next.

With this representation I do not mean to imply that an activity is situated within only one practice, nor that the relationship between activity and practice is subordinate, with the practice "higher than" or "surrounding" the activity. An activity will negotiate its usage with any number of practices, depending on where we draw their boundaries, and they may all negotiate with each other as equals in what Hutchins calls "adaptive interaction among subsystems" [35].

Instead, I mean the different terms "activity" and "practice" to reflect their different roles *relative to a particular design.*

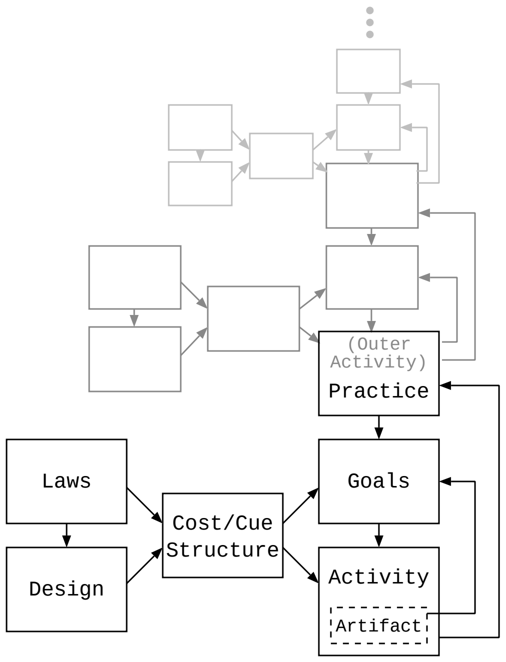


Figure 1.14: A multi-level design ecology. Activity at one level serves as the practice for another.

1.2.6 Revisiting the Design Cycle

To come (literally) full circle, let's reinstate explicitly in the model the design cycle's closure, the feedback on which the designer reflects. In Simon's *satisficing* model (Figure 1.6), that feedback is a combination of the satisficing criteria and the steering mechanism. In our expanded model, that feedback is represented in Figure 1.15.

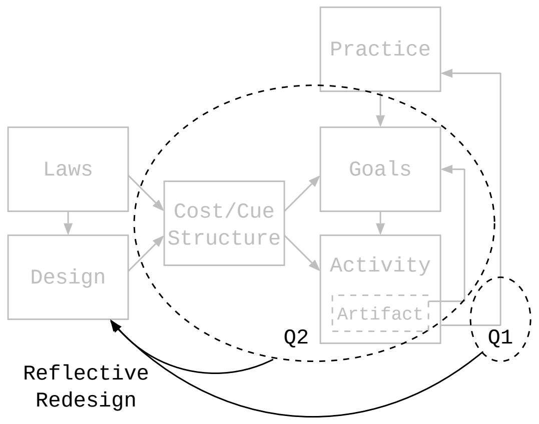


Figure 1.15: The designer's reflection steers redesign.

Our original two questions have evolved as follows:

Q1: When is a design good?

$\longrightarrow$ Assuming a user's goals are fixed, does the design satisfice?

$\longrightarrow$ Does the *emergent cycle of goals and activity* satisfice?

$\longrightarrow$ Is the emergent activity *satisficiently compatible* with the surrounding practices?

Q2: How do we make it that way?

$\longrightarrow$ What mechanism will steer redesign?

$\longrightarrow$ How is activity shaped by the user's reflective cycle and the design's cost structure?

Note that we've never answered Q1, which still includes an ill-defined satisficing criterion. We've only pushed the problem to a different part of the ecology, away from the design *per se* to the interaction of the activity and its practice. But in doing so, we've pushed it out of the way of Q2, beyond the region of the ecology where the design has the most direct impact. That is, before we need to make any evaluation of whether our design is "good", we can learn a great deal about the system of influences we'll have to manipulate indirectly through the design.

The goal of this thesis is not to answer question Q1, to decide whether Dynapad is a "good" design. That would require longitudinal observations to understand how Dynapad is (or isn't) adopted by users in their work practices. Although ideal, that evaluation is beyond the scope of this work.

The goal is instead to make progress on question Q2, to explain why Dynapad induces the shorter-term reflective cycle of activity that it does. To do so, we must examine both ends of the cost structure: how the design, subject to certain inviolable constraints, induces costs and offers cues, and how those costs and cues influence the reflective cycle.

Dynamics of the Ecology

The dynamics of this ecology arise from the presence of three loops:

- At the shortest time scale is the user's *reflective cycle of sensemaking*: each action modifies the artifact (e.g. the Dynapad workspace), triggering new interpretations and discoveries which push the activity in a new direction. This cycle is significantly impacted by the tool's cost structure.

- At a longer time scale is the user's *negotiation of usage*: as she grows to understand the tool's abilities, she potentially changes what she uses it for. This cycle is significantly impacted by the user's outer environment, the daily context in which she operates.
- At multiple time scales is the designer's cycle of *evaluation and redesign*: as his understanding of the ecology grows, he modifies the design, and thereby the cost structure, to nudge the user's activity in a helpful direction.

  Ideally, these modifications could reflect an answer to Q1: whether the activity is compatible with the desired practices, as revealed by the *negotiated usage* cycle. For this purpose, the redesign cycle must have an extremely long time course in order to observe longitudinally that usage cycle.

In the end, the designer's process seems to have essentially the same structure as the simple "Generate and Test" paradigm. But treating the process as an ecology, represented by Figure 1.15, acknowledges three key differences:

- The purpose of the design, the standard by which its effectiveness is measured, is a moving target.
- The object of evaluation is not a measure of utility or goodness but the *activity* induced by the design.
- The evaluation is not quantitative but primarily qualitative. It seeks to understand not just that one design is better than another, but *why*. That explanation must be expressed as an interconnected system of influences, itself an ecology.

1.3 The Road Ahead

This work explores the ecology of a design: how the cost structure of an artifact shapes behavior, and how consideration of this impact suggests alternative designs. Its

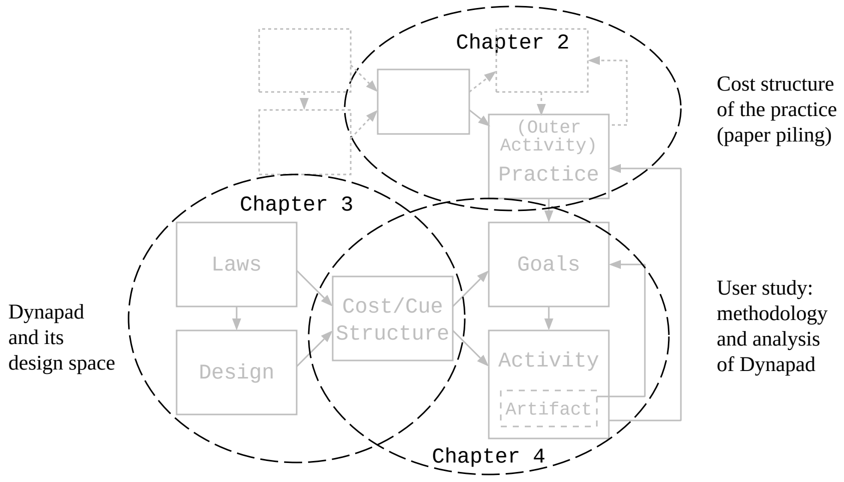

Chapter 2
(Outer Activity)
Practice
Cost structure of the practice (paper piling)
Chapter 3
Laws
Design
Cost/Cue Structure
Goals
Activity
Artifact
Dynapad and its design space
User study: methodology and analysis of Dynapad
Chapter 4

Figure 1.16: The chapters ahead correspond roughly to these three subsystems of the ecology.

organization approximates the structure of one iteration of a design cycle, which has roughly three parts, summarized in Figure 1.16.

Envisioning: Chapter 2 establishes the general goals of Dynapad by drawing on the existing practice of managing paper archives. It also examines the cost structure of that practice, which will serve as a foundation for understanding the cost structure of Dynapad.

Designing: Chapter 3 presents the details of Dynapad's design, including the laws, trade-offs, and historical precedents that constrain it. Although the design has evolved through several iterations, we'll focus on a version or two of the design family, and situate them in a history of earlier iterations.

From these details, we'll see how Dynapad changes the cost structure of paper management.

Evaluating: Chapter 4 describes how that cost structure actually plays out in the activity of two users. This will expose some unforeseen aspects of the cost structure and suggest design modifications to alleviate certain breakdowns in the activity.

2

The Practice and Cost Structure of Managing Paper Collections

In Chapter 1, we've seen that the interactive behavior that emerges in a designed environment is the result of both the cost structure of that environment and the momentum of an existing practice. This practice includes not only a set of habits but also explicit and implicit goals and expectations a practitioner brings from it. And of course, these goals, habits, and expectations are themselves shaped by the cost structure of that ancestral practice.

The goal of this chapter is to characterize the practice of managing paper[1] archives which Dynapad inherits, identify the goals and expectations inherent in this practice, and illuminate how these emerge from its own cost structure.

Piling vs. Filing in Paper Archives. A seminal work in understanding how people manage collections of information is Malone's "exploratory observation" of office workers' organizational strategies [49]. Malone identifies *filing* and *piling* as the two primary strategies for organizing paper documents. Roughly characterized, filing is a "neat" strategy of classifying documents systematically into storage cabinets, and piling is a "messy" strategy of leaving documents in piles around one's workspace. Malone's key insight is that piling is not merely a breakdown of organization, but an adaptation which

[1]The particular affordances of paper, as opposed to digital documents, have been examined in detail by Sellen and Harper [66].

offers unique advantages over filing alone.

2.1 Piling as a Model Practice

Before exploring the details of piling, let's consider why it's an appropriate model practice to draw from. Lansdale cautions that piling *per se* should not be a design goal:

> The piles that Malone reports are not, in a simple sense, representative of a need in the user. Quite the reverse, in fact. They are a compensating strategy for the problems of classification. [46, p.56]
>
> It might seem straightforward to suppose that we can translate observed strategies of information handling from existing paper-based methods to computers.... In principle, however, this must be a mistake... No one would suggest the introduction of unstructured 'piles' of documents in a computer environment... [46, p.56]
>
> Concentrating upon piles is to caricature what happens when procedures from office practices are reapplied to computers... [46, p.57]

Lansdale's core concern is valid: if the strategy of piling is merely a side effect of the cost structure of managing paper archives, such that it disappears in a changed (e.g. digital) ecology, we shouldn't sanctify it when designing that ecology. Instead, we should work more directly to improve the cost structure which induces it.

But of course, he's also partly mistaken: subsequent work [50], including this, has indeed introduced such piles into a computer environment, where they have remained useful.

Even if we were to do no more than to implement piles literally as Lansdale describes, we have reasons to expect some utility from them. First, users come to the computer not as blank slates but as experienced pilers, who expect to reply on piles at times. The computer should support not just their direct needs but also their habits and expectations from managing physical collections, even if those habits are vestigial. Second, the challenges of information management which lead to piling — including but not limited to categorization — may be mitigated on a computer but probably won't go away. Unless we drastically restructure the ecology (which is risky and disorienting), practiced coping strategies should retain some value. And third, piling is not merely a compensatory by-product, imperfectly serving rigid information-retrieval goals. It is a legitimate activity in its own right and introduces its own goals, as we shall see.

Furthermore, we are not limited to implementing piles literally. One of the goals of this work is to *dissociate* various attributes of piles in order to *selectively engineer* affordances that contribute most to a congenial ecology. In other words, we can *extend the metaphor* of what a "pile" is to include structures more appropriate to the cost structure of a computer environment.

And of course, whether we implement any variant of piling or, as Lansdale suggests, merely attempt to mitigate in software the difficulties which induce piling with paper, we must first understand those difficulties. Section 2.2 ahead attempts to characterize the differences between filing and piling, as observed by Malone and others, and to connect the properties of each with the costs they impose and advantages they afford.

The Role of Exploratory Observation

Malone's self-described methodology is one of "Exploratory Observation". His intent is not to reach quantitative or definitive conclusions but rather to reveal the domain's essential qualitative phenomena and the connections between them.

Extending Malone's work, Whittaker and Hirschberg [70] focus on the role of personal *archives*, information that people maintain over relatively long time periods. Their own methodology is more quantitative than Malone's, and they observe statistically significant trends among their participants.

Nevertheless, for our purpose here, both Malone's and Whittaker's contributions play the same role. Whittaker's results, while internally valid, may not hold externally, in a different community of participants or in a changed, computer-based ecology. But like Malone's observations, their value is to identify productive units of analysis — behavioral strategies, hazards, and affordances — and examine potential interactions between them.

2.2 The Cost Structure of Piling and Filing

To begin, let's examine how Malone distinguishes "filing" and "piling". Both strategies group elements (documents) into larger structures (e.g. folders, stacks). Malone avoids an explicit definition of their difference but appeals to our common-sense understanding of the terms. *Files* are *formal* and "neat", explicitly labeled and systematically ordered (alphabetically or chronologically, for example). *Piles* are *casual* and "messy", unlabeled and often unsorted, both within and between piles; any organization is implicit.

We can also think of differences in their physical structure and placement: files are typically enclosed in folders within drawers, and piles are unbound and stacked on a flat surface. Malone seems to consider these differences to be non-essential (though not irrelevant, as we shall see). For Malone, the essential difference concerns the *organizational system.* For example, he considers books to be "piled" on a shelf, even when unstacked, if they are placed in groups but not sorted in an explicit way.

Hybrid Structures. Some organizational structures are ambiguous. For example, "In-" and "Out-" boxes may be explicitly labeled and neatly bounded (i.e. boxed) like files, but stacked vertically within the workspace like piles. They also play a cognitive role often taken by piles, as a temporary staging area prior to further evaluation.

Such hybrid examples defy classification in Malone's dichotomy. But our goal is eventually to surpass it and instead to characterize various structures as combinations of dissociable features whose cognitive affordances are brought to light by contrasting piles and files.

Mixed Strategies. Although Malone divides his subjects into "messy" pilers and "neat" filers, he recognizes that the two strategies are often used together, and we can infer than most of his (and Whittaker's) subjects used both to some degree. Malone's concern is to identify the qualitative properties of a system in which either piling or filing *predominates.* I will take the same approach in the discussion ahead.

It is possible that the differing strengths of filing and piling combine linearly in a mixed strategy. But it seems more likely that these strategies interact in non-linear ways. A well-integrated mixture of filing and piling may be better in most respects (or a bad mixture worse) than either pure strategy. Our purpose is to identify the likely contributions of each strategy to the system as a whole.

2.2.1 Piles as Bundles

Our default image of a pile is a *stack*. But adopting that model from the outset makes unnecessary commitments to details of physical structure. In keeping with Malone's loose definition, for now let's consider piles only as indeterminate "bundles" of items and attribute to them only *environmental* attributes, regarding their *role in a system* of organization and activity.

Exposure: Visibility and Accessibility

We've identified piles' first environmental property as *implicit organization*: piles are unlabeled and often unsorted. But besides this difference in organization, piles and files have another essential difference (which Malone leaves implicit): files are typically stored compactly out of one's immediate workspace (in a drawer for example), while piles are left *exposed*, lying in the open. This is an example of a differential cost-structure of information in the sense intended by Card et al. [14]: piles have a generally lower physical access cost than files. But exposure reduces cost in two ways: piles are easier both to *reach* and to *see*. That is, they have both greater *accessibility* (reduced cost of *interaction*) and greater *visibility* (reduced cost of *perception*). Or in the terminology of Chapter 1, accessibility improves *cost-structure* and visibility improves *cue structure* [43].

But under what conditions are these costs reduced? If one is already engaged in filing, already looking in a file drawer, needed files can be very easy to see and access. The access cost of piles is less primarily when *switching* to document storage or retrieval *from some other activity*. Another way of characterizing piles' exposure is that the space used to organize and interact with them is *shared for other purposes*. Piles' visibility means they can be seen not only when storing and retrieving documents but when looking for one's keys, reading mail, or eating lunch. And piles' greater accessibility also means they can be disrupted by such activities.

Although Malone [49] and Whittaker & Hirschberg [70] are well aware of these differences and illustrate them in several explanations, they never explicitly acknowledge *exposure* as a property of piles independent of their *implicit organization*. Especially for purpose of engineering a digital analog of filing and piling, it is important that we understand not just *what cognitive effects* differentiate the two structures, but *what properties* give rise to those effects. Our goal in this section is not only to summarize the effects reported by Malone and Whittaker, but to clarify which properties of files and piles contribute to these effects. Dissociating organization from exposure is a first step.

An Example of Competing Influences: Retrieval

Let's pause to look ahead at the one of the goals both piling and filing try to satisfy: *retrieving* information from a collection. If we consider retrieval to be, in Lansdale's terms, a primary need of the user, and filing and piling to be adaptive strategies toward that end, which strategy is ultimately better at supporting retrieval?

On one hand, we can expect the investment in files' organization to pay off, making particular documents easier to locate when needed. On the other hand, Whittaker reports:

> Filing does not always guarantee easy access to information...With complex data, filing systems can become so arcane that people forget the filing categories they have already created, leading to duplicate filing categories. Accessing only these duplicates leads to incomplete retrieval, because information in the original files gets overlooked. [70]

Malone and Whittaker agree: neither strategy is objectively and consistently better. As a measure of retrieval ease, Whittaker estimates the overall number of documents retrieved by pilers and filers and concludes that they are not significantly different.[2]

Not only do the strengths of piling and filing differ, they often *compete*, as with retrieval. If we measure only their net (and often indeterminate) effect, we fail to appreciate how their affordances contribute differently and interact.

Keeping Score. As we consider in this section the various cognitive consequences of piling and filing, we'll keep track of our progress by incrementally building an "influence

[2]Although the overall volume retrieved by pilers and filers was the same, it was a significantly greater *proportion* of piled archives because they were generally smaller; see 2.2.1.

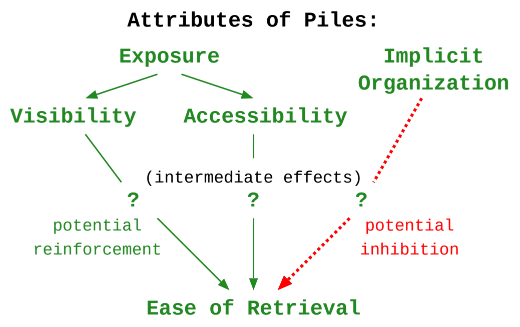


Figure 2.1: An influence graph: attributes of piles and some potential cognitive implications.

graph", as Figure 2.1 suggests. It is intended not to be a predictive or exhaustive model of archive management but merely an *accounting of potential influences*. This representation has three purposes:

1. to clarify the dissociations between attributes (e.g. organization vs. visibility vs. accessibility);

2. to clarify the contributions of each attribute to the overall cognitive ecology (e.g. what *about* piles leads to smaller archives?);

3. to recognize indirect, complementary, contradictory, and reciprocal effects.

For example, we've suggested that there are contradictory influences on retrieval; soon we'll see why in detail and identify a set of attributes that contribute to it.

Mechanical Costs of Piling

As Malone observes, piling has lower mechanical (as opposed to cognitive) costs than filing for two reasons. The first is a consequence of piles' accessibility: piles, already in the workspace, are immediately in reach; no drawers, folders, or other containers need to be opened. This also makes retrieval easier.

The second cost reduction results from piles' implicit organization: piling an item eliminates the effort of obtaining, labeling, and integrating new folders.

Easier categorizing means easier *re*-categorizing, making the organizational system more flexible and thus more likely to remain appropriate and effective as one's relationship to the collection changes. (We'll consider this effect later in 2.2.1.)

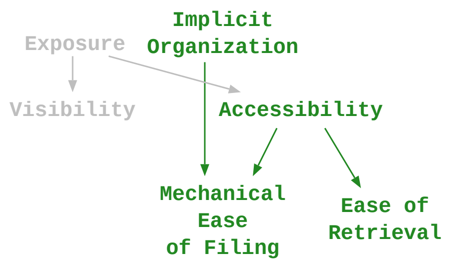


Figure 2.2: Being both implicitly organized and immediately accessible, piles have lower mechanical costs than files. Piles' accessibility also helps retrieval.

Reminding

Malone emphasizes the difference between retrieving or *finding* and *reminding.*[3] Finding an item requires prior intent and cognitive recall; discovering an item requires only recognition. Piles are better than files at reminding us to do things. As one of Malone's subject says, "if I don't put it where I can see it, I won't do it" [49].

More generally, we use external structures to mediate our activity [35]. For example, forms help us remember how to carry out certain procedures (e.g. purchasing) by directing our attention through the constituent subtasks [11].Malone suggests that filers tend to have more highly structured jobs than pilers [49, p.106]. If this is indeed the case, there are two reasons to expect it. First, the well-defined structure of "routine" jobs may make it easier to create and maintain an effective filing scheme. But additionally, workers with less well-defined practices may rely on piles as a mediating structure, reminding them how to proceed.

Piles are better at reminding than files because of their visibility: they can be seen when one is doing an unrelated activity in the same space. They remind because they *share visual space* with practices unrelated to retrieval. Reminding happens when *separate practices interact* because their resources occupy the same space.

[3]Barreau and Nardi [2] and a critical response by Fertig et al. [23] have also examined the importance of reminding in computer file systems.

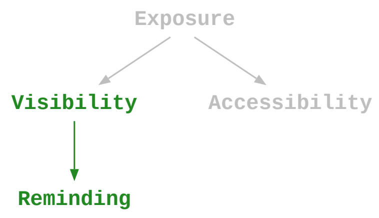
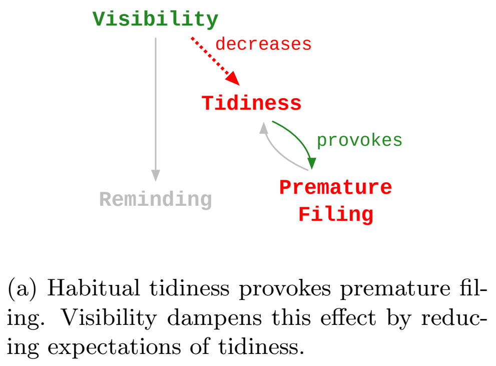


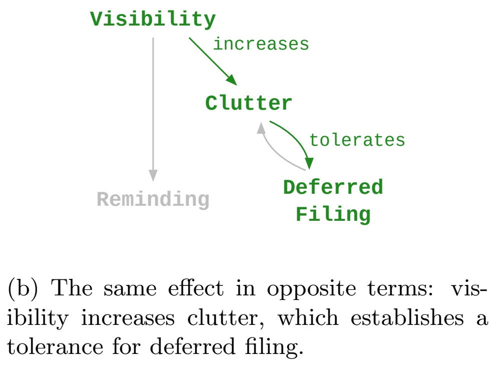
Figure 2.3: Piles' ability to *remind* results from their greater visibility.

Premature Filing and Archiving

Whittaker and Hirschberg initially assumed that pilers would tend to accumulate more useless information than filers. But among their participants, the opposite was true: pilers tended to have *smaller* archives. Whittaker's explanation centers around *premature filing*. Those accustomed to filing, thereby keeping tidy desktops, feel some obligation to classify and file new documents immediately and often prematurely. Conversely, piles increase visual clutter[4], and that precedent for clutter can increase one's tolerance to defer filing new documents until their value and role is better understood.



(a) Habitual tidiness provokes premature filing. Visibility dampens this effect by reducing expectations of tidiness.



(b) The same effect in opposite terms: visibility increases clutter, which establishes a tolerance for deferred filing.

Figure 2.4: Premature vs. deferred filing

Considered in isolation, there is the potential for a self-reinforcing cycle, a habituation to clutter: the more one defers, accumulating clutter, the more normal clutter seems and the more one defers. Of course, this cycle may also correct itself if growing chaos sufficiently increases pressure to clean up.

[4] "Clutter" is meant to imply only visually distracting bulk, not necessarily disorder. A desktop full of immaculate piles is still cluttered in this sense.

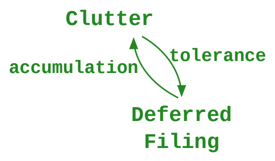


Figure 2.5: Habituation to Clutter. Increasing clutter and deferral can be mutually reinforcing.

Reminding allows deferral. In addition to the effect of clutter, filing is more easily deferred when one is assured of being reminded (Figure 2.6). In the words of one of Malone's subjects:

> "You don't want to put [a pile on the desk away] because that way you'll never come across it again... Leaving them out means that I'm going to come across them again, and at that time... I'm going to decide what to do about them." [49]

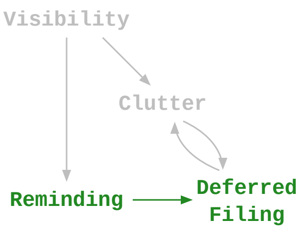


Figure 2.6: Reminding ensures later opportunities to file.

Deferring Effort. So far we considered only ways in which piles, by their visibility, reinforce deferred filing. But an additional influence counters this trend: the mechanical effort of filing also induces filers to defer (Figure 2.7(a)), and piles therefore reduce this effect. That is, increased mechanical ease inhibits deferral (Figure 2.7(b)).

Deferred Filing
Mechanical Difficulty of Filing
Deferred Filing
Mechanical Ease of Filing

(a) When filing is difficult it is deferred.

(b) Conversely, the ease of piling reduces the need to defer.

Figure 2.7: The mechanical cost of filing and deferral

For example, one of Malone's subjects

> had an elaborate (and presumably useful) system of cross-reference sheets in his... files, but... he had many documents scattered around his office waiting to be filed. [49, p.107]

Apparently the difficulty of maintaining this complex filing system created considerable pressure to defer filing and pile documents instead.

Archive Size. An obvious consequences of deferred filing is that items accumulate in the collection. When one's archive comprises piles, more piling enlarges that archive. However, Whittaker observes some opposite forces by which deferral keeps archives small.

First, premature filing *demotivates* clean-up. Filers are reluctant to discard information they have invested effort in organizing:

> Once you've committed something to paper, and it sits in a file, it requires a separate act of attention and will to take the stuff back out and reduce files to their bare minimums. [70, p.162]

Deferred filing eliminates this disincentive.

Second, pilers allow themselves more and better-informed *opportunities* to re-evaluate and discard information. Deferred filing decisions can prevent low-value items from entering the archive, and reminding can prevent them from invisibly persisting. As Whittaker notes:

> ...pilers, unlike filers, encounter and discard extraneous information while searching for other information. [70, p.164]

Such *opportunistic pruning* helps reduce the size and increase the value of archives.

Finally, the tendency of important items to float to the top of piles (see 2.2.2) offers a convenient *heuristic* for cleaning out archives: periodically sweep obsolete items off the bottom of piles.

Organizational Instability

Let's return to the effects of accessibility. Because piles share space with other activities, they can be unintentionally disrupted and the information encoded in their organization lost:

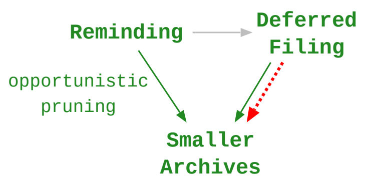


Figure 2.8: Piling can yield smaller archives by reducing premature filing and creating frequent opportunities for pruning.

> Taken to excess... piles can dominate not just working surfaces but all areas of the office. In clearing floors, tables, and chairs of their accompanying piles, in order to use them for their normal purposes, urgent piled information can be lost. [70]

That is, accessibility threatens the stability of organization.

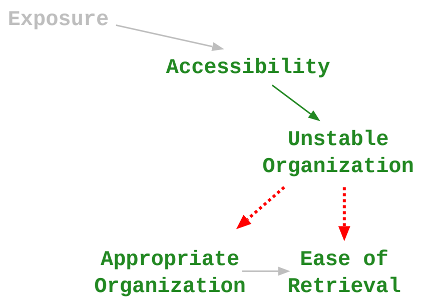


Figure 2.9: Piles' accessibility leaves them susceptible to unintentional disruption; their organizational scheme is potentially unstable. The resulting organization is inappropriate and retrieval is harder.

Not only unrelated activities but also *pragmatic* piling actions can disrupt piles' organization. For example, if I expect to access several items from a pile across the table, I might move the whole pile closer to make it even more accessible. But doing so may obscure information reflected in its original position — for example, that it belongs to a meaningful set of adjacent piles.

Such instability erodes the effectiveness or *appropriateness* of the organization (see 2.2.1) and consequently the ease of retrieval.

Organizational Clarity

The primary penalty of organizational instability is losing one's *understanding* of how the piles are organized. Physical disruption makes the organization unclear. Here is the potential for another vicious cycle: the less well understood the organization, the greater the chance of misplacing or misclassifying items, and the more unstable the organization.

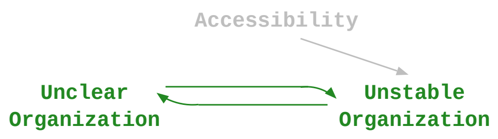


Figure 2.10: Chaos from disruption.
An unstable organization can quickly become misunderstood. Misunderstanding can lead to misfiling, further disrupting the organization. The more unstable the organization, the faster it becomes unclear.

Organizational clarity is impacted by factors besides disruption. Piles' implicit organization especially hinders clarity: the absence of labels not only prevents verifying one's understanding of a pile's meaning but also permits meanings to remain inchoate and inconsistent. Although this can offer the advantage of greater flexibility (see 2.2.1), it may also contribute to misinterpretation of the organization.

Cognitive Difficulty of Filing. While implicit organization reduces the mechanical cost of piling items, the resulting potential for confusion increases the *cognitive* cost of piling. This may create a tendency to defer categorization.

Visibility Helps Clarity. Visibility amplifies the cue structure of the organization. Whatever information is encoded in the spatial layout of piles, when visible, can be used to verify one's understanding and perceive existing categories. Visibility reduces the cost of *glances* which verify and refresh one's mental picture of the organization.

Consider again Whittaker's caution about files:

> With complex data, filing systems can become so arcane that people forget the filing categories they have already created, leading to duplicate filing categories. [70, p.161]

This cognitive disconnect occurs because files' organization is less visible.

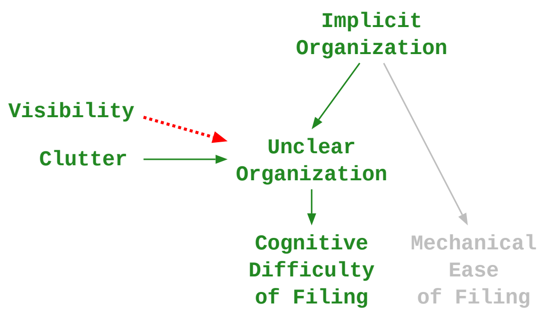


Figure 2.11: Piles' implicit organization, compounded with visible clutter, can obscure the organization of the system. So piling makes it potentially more difficult *cognitively* to classify items, even when *mechanically* classification is easier. However, piles' *visibility* can make the organization more clear.

Clutter Hurts Clarity. The advantage of visibility is offset by its inevitable byproduct, clutter (i.e. visual bulk and complexity). Visually complex organizations can become overloaded with interpretations, making it more effort to understand the intended organization.

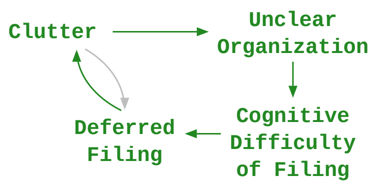


Figure 2.12: Chaos from overload.
Deferral and misunderstanding contribute to a vicious cycle of disorganization.

Here is yet another potential vicious cycle. The earlier cycle of habituation to clutter and deferral is made worse by the additional influence of unclear organization. Deferring can add clutter, and clutter can obscure organization, making it cognitively more difficult to classify and thus increasingly tempting to defer incoming items. This cycle may be one facet of the general problem of *overload*.

Organizational Flexibility

The effectiveness of an organization system depends in part on whether it's structured *appropriately* for the ways it's used. For example: is the vocabulary used for retrieval the same used for categorization? Does the overall structure of the organization illuminate the structure of one's work?

Even if an organization system is appropriate initially, as our work (or our understanding of it) changes, that system may become inapt and inefficient. It takes time to understand the flow of one's own information management, and that practice may be continually adapting. Therefore we can expect that any particular organization may become inappropriate or obsolete over time, To mitigate this, a system should be *flexible*. That is, the cost of restructuring the organization should be low.

Appropriateness is not the same as understanding: one may understand perfectly an organization which is nevertheless inapt and ineffective. And flexibility is not the same as instability; flexibility is the *deliberate restructuring* rather than accidental and possibly invisible disruption.

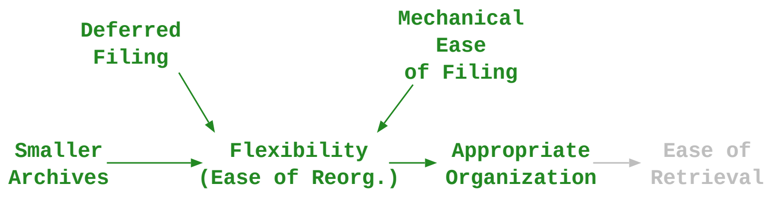


Figure 2.13: Small archives, deferred filing, and mechanical ease of re-categorizing all afford the organizational system greater flexibility. This helps the organization stay appropriate to the flow of work and the structure of information, and the more apt organization facilitates retrieval.

Mechanical Ease Helps Flexibility. We've seen that implicit organization both eases the mechanics of filing and tolerates ambiguity in categorization, which can hinder clarity. But these qualities also make the organization more flexible. For example, it requires no effort to change the meaning of an unlabeled pile. It takes minimal effort to combine piles, or to move them to areas with established meanings. Piles' embrace of ambiguity combined with their low cost of re-categorizing makes them more responsive as organizational needs change.

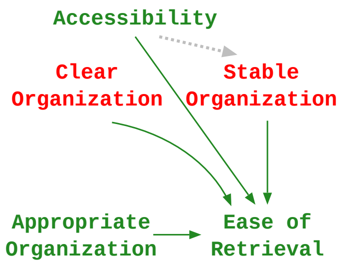
Smaller Archives Help Flexibility. The smaller a collection is, the easier it is to reorganize.

Deferred Filing Helps Flexibility. Besides shrinking archives, deferral also helps flexibility more directly. Whittaker observed that premature filers were reluctant to throw away information that they went to the work of filing. Similarly, we might expect a reluctance to *break up* existing categories to *re-categorize* the items, especially for frequently-accessed and well-defined groups. If an organization is well-understood and the cognitive effort of categorizing is low, the path of least resistance is to reinforce the existing categories even if they are counter-productive. Ironically, by increasing the cognitive difficulty of categorization and increasing deferral, piling may slow the reinforcement of an ineffective system, thus helping flexibility.

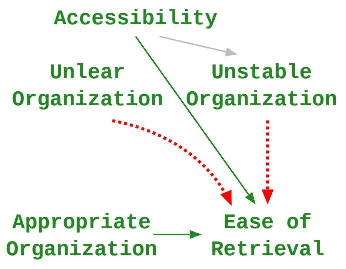
Finally, because a flexible system stays more appropriate, it helps retrieval.

Retrieval, Redux

As predicted earlier, piling has both advantages and disadvantages for retrieval. Now we can see why in detail.



(a) Positive (but incompatible) influences on retrieval.



(b) Piles' hinder stability and possibly clarity.

Figure 2.14: Piles' net influence on retrieval ease is indeterminate.

We've exposed four positive and direct influences on retrieval: accessibility, plus an organizational system which is stable, appropriate, and clearly understood. We might infer from the network of indirect influences that piling generally improves accessibility and appropriateness, but generally hinders clarity and stability. Thus the overall net effect of this conflict on retrieval is unpredictable.

Figure 2.15 summarizes the influences we've considered so far.

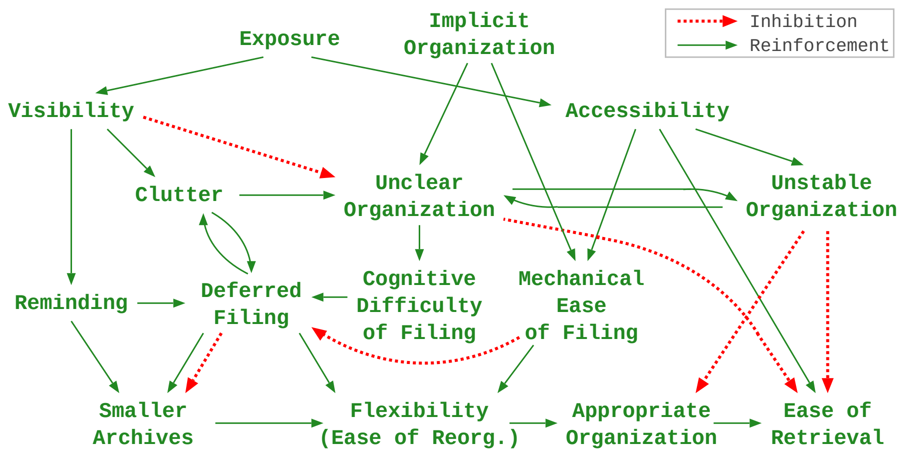


Figure 2.15: Summary of potential influences of piles' organization and exposure.

2.2.2 Piles as Stacks

So far, we've considered a largely abstract interpretation of piles, distinguished only by their implicit organization and exposure to the workspace. Both of these properties concern the role of piles as abstract "bundles" within a system of bundles, without regard to their *internal* structure. Most of the influences above should apply equally well to non-literal "piles", such as Malone's example of books grouped on a shelf.

But of course, the most salient and ubiquitous form of a pile, especially for paper, is the *stack*. Next, let's consider piles as stacks and explore how their internal structure interacts with the effects we've discussed already. Figure 2.16 previews and situates the upcoming additions to our influence graph.

Stacking and Substructure

The defining property of a stack is *serial occlusion*: each item occludes, visually and physically, the item below, such that only the topmost item is directly visible and accessible. Lower elements can be reached, but only by moving the ones above. The further down the stack the target lies, the more effort is required to reach it. So occlusion amounts to a top-down *differential negation* of visibility and accessibility. Piles are generally more visible and accessible than files, but occluded piles (i.e. stacks) have these advantages somewhat neutralized.

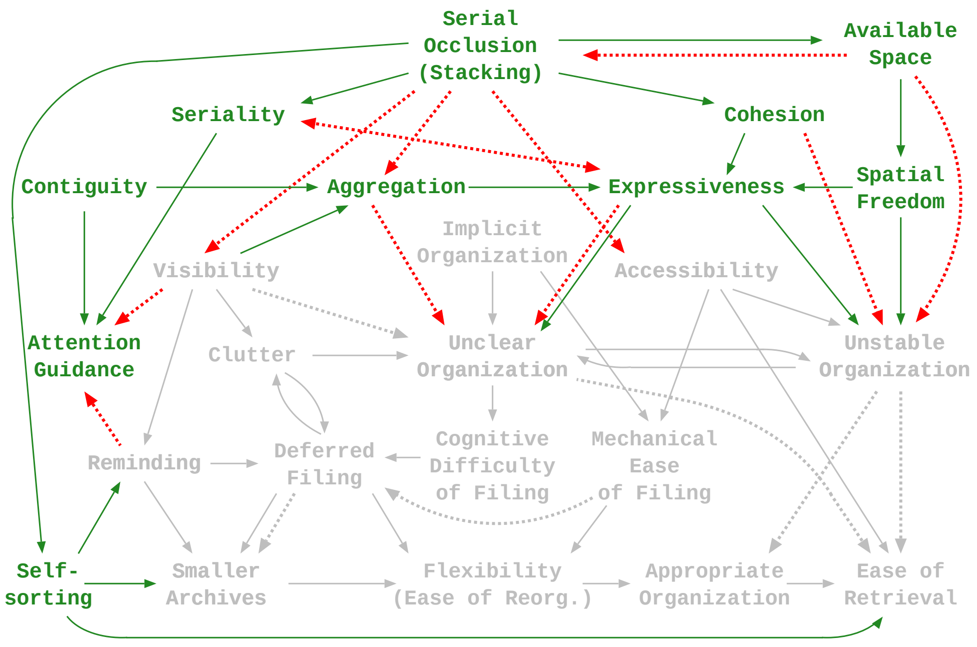


Figure 2.16: Preview of additional influences involving piles' internal structure

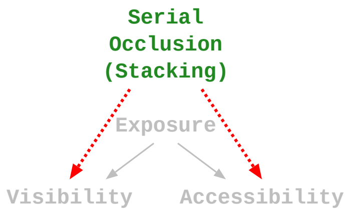


Figure 2.17: Stacking occludes all but the topmost items, reducing overall visibility and accessibility and differentially negating exposure.

Note that occluded items can still be partially visible at their edges, especially if piles are "ragged" (imperfectly aligned) or if items have sufficient thickness to be identifiable from the side.

Self-sorting. One helpful byproduct of serial occlusion is that stacks can exhibit *self-organization* through our interaction with them. In the simplest case, imagine adding items to a stack over time without retrieving any. Because it takes extra effort to insert items within the stack, we naturally add them to the top. So the stack preserves a historical record of the interaction in the sequence of its members.

When we eventually retrieve items from within the sequence (which, due to occlusion, requires extra effort), and replace them on top, the long-term result is that important (frequently-used) items float to the top and little-used ones sink to the bottom.

This has several advantages. First, it optimizes the access-cost structure for retrieval: the most-retrieved items have the lowest cost, so the average cost is minimized. Second, important items on top serve as reminders (see 2.2.1). Third, it facilitates clean-up: obsolete items can be "swept" off the bottom of piles (see 2.2.1), keeping archives trim.

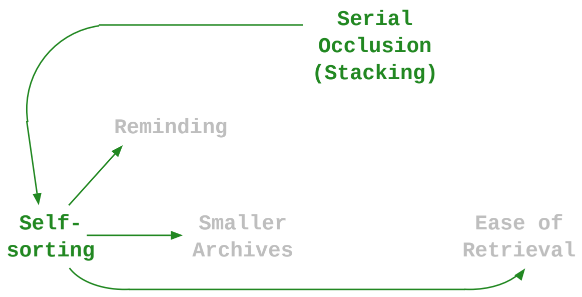
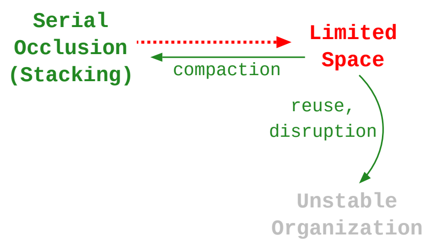


Figure 2.18: Piles' top-first accessibility tends to sort them by usage: frequently-used items float to the top. This helps retrieval, reminding, and clean-up.

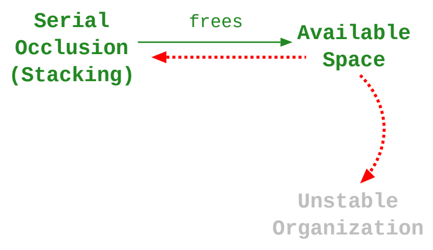
This self-organization *could* also be true of files, which we can think of as sideways stacks within folders. If we consistently access files front-first, they emulate stacks. But they just as easily afford dropping items in randomly, or turning around to be accessed *back*-first. So files require a degree of discipline to match what piles regulate naturally.

Space Limitations. An obvious advantage to stacking is that it preserves space, which is usually a limited resource. The more limited the space, the greater the pressure to stack items and combine piles. Such shuffling can disrupt the organization, thereby increasing instability. But fortunately this is self-dampening cycle: stacking frees up space and reduces the pressure for further stacking.

Cohesion. Another important side effect of stacking is *cohesion*: even without a wrapper, the friction and compactness of stacked items makes it easy to pick up and move a stack without disrupting its structure. This is true for files as well, which also have a



(a) Limited space forces stacking and displacement



(b) Stacking frees more space

Figure 2.19: Competition for available space

protective wrapper. In contrast, a loose heap of papers probably falls apart when moved.

Stacks' robust portability reduces the impact of potential disruptions. Moving a stack out of the way, for example, may lose information reflected in its position but will preserve information within its structure. Thus cohesion reduces organizational instability.

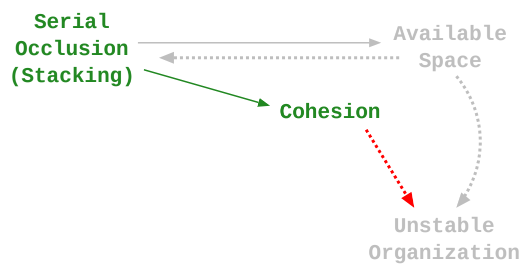


Figure 2.20: Stacking makes piles (and files) cohesive and portable, reducing the effects of disruption.

Spatial Freedom and Expressiveness

As we've seen in the example of self-organization, both piles and files can encode information in their *substructure*, the way their members are physically arranged. At minimum, they each afford one ordinal dimension of encoding (as with stacking), and more complex structures are possible through nesting: files can contain subfolders, and piles can include "subpiles" delimited by physical markers (e.g. folders), offsetting, or rotation. Stacks can be turned sideways, enclosed in a folder, and "filed". So at first glance, it seems that both structures have the same *expressiveness*, the ability to encode information.

But piles are more expressive because they have greater *spatial freedom*, both within and between groupings. Meaning can be expressed by position and spacing, and generally speaking, files are *discrete* but piles afford *continuous* placement. For example:

- Within files, items cannot reliably be offset or rotated. Items sink to align their bottom edge with the folder's spine, and cannot offset sideways without extra-wide folders. They cannot rotate except by 90 degrees, and even then may not fit in their drawer. Within piles, items have greater flexibility of position — for example, jutting out for greater visibility even when not on top, or rotated as a marking.

- Unlike files, piles preserve empty space between them. This allows visual gestalts like *proximity* of related piles and *continuation* of linear arrangements. It also allows linear scaling to depict quantitative values.

- Membership in piles can be qualified. Dubious items can be placed near a pile instead of in it, and ambiguous items can straddle adjacent piles or lie midway between more distant piles. With files, elements are either in or out and cannot be shared without copying.[5]

In summary, piles' spatial freedom can be used to simplify perception [42] and optimize the cue structure [43] of collections.

Freedom requires space. Free space does not mean spatial freedom; one can have lots of empty file drawers, all limited to discrete positioning. But utilizing continuous space to encode information requires a certain minimum of available space. Gestalt grouping requires whitespace around the group, and scaled (as opposed to ordinal) linear layouts require whitespace in their gaps. Nearness is only relative, and crowding from limited space will obscure it.

[5]The need for *multiple classification* arises in both piling and filing, and is adequately solved only by duplicating items. This cost is presumably the same in both strategies.

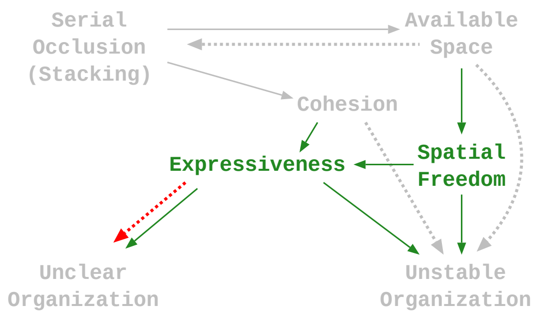


Figure 2.21: Expressiveness allows complex representations both within and between piles. Information within the structure of a pile is protected by cohesion. Complex representations have the potential both to improve and degrade clarity, and are more fragile (unstable).

Freedom hurts stability. Once space is used to encode information, the organization is more sensitive to disruption. For example, if deliberate misalignment is used to mark piles, then displacing a pile accidentally or for pragmatic reasons loads it with unintended implications.

Cohesion helps expressiveness. In the same way that it helps stability, cohesion helps preserve expressive arrangements within groups. And by reducing the cost of pile movement, it reduces the cost of creating and adjusting inter-group arrangements.

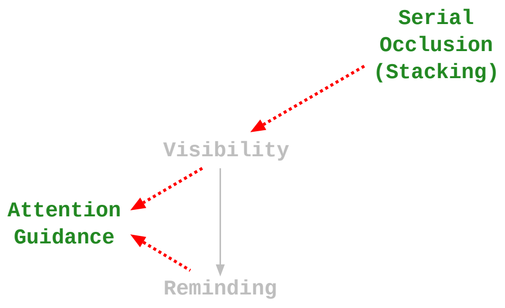
Expressiveness both helps and hurts clarity. Carefully arranged workspaces can help to keep organization clear by optimizing cue structure [43]. But there is a danger of overdoing it, putting in so many spatial relationships that organization becomes confusing.

Expressiveness hurts stability. A carefully arranged workspace which is saturated with meaning is also fragile and susceptible to disruption.

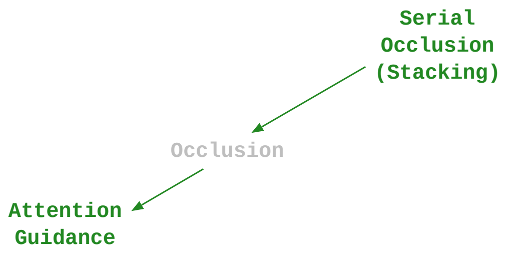
Visibility and Attention

Occlusion, like exposure, has facets of visibility and accessibility. We've examined some consequences of occluded access; now let's consider occluded visibility. So far we've portrayed visibility as mostly good. Its most negative consequence so far has been *clutter*, and even that has some potentially beneficial effects (e.g. deferred filing). But we've not yet discussed the most serious cost of visibility: it overloads our *attention*.

Reminding happens when our attention is *distracted* by visible items. By reducing visibility, stacking reduces distraction and helps us regulate our attention.



(a) Stacking limits visibility, which creates distraction.



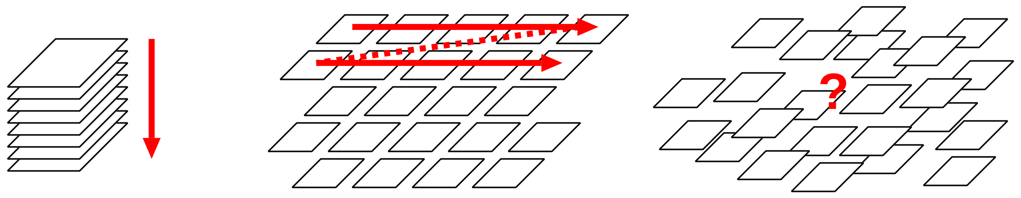
(b) In positive terms: stacking guides attention by occlusion.

Figure 2.22: Reducing visibility helps guide attention.

Serialization. Stacking helps guide attention in another way: it *serializes* items in the group to make it easier to iterate attention over them. Imagine that you're looking for a particular item in a large, unsorted pile. The most efficient strategy is to examine each item once and only once until we find the target. But this procedure requires some *metacognitive* effort [43] to plan and execute. That effort is minimized when there is a serial "path" through the set that visits each item once. A stack is perfect: each item has only one successor, and it requires no cognitive effort or memory to follow the sequence. Put another way, a serialized arrangement eliminates decision points in the problem space [42, 43]. The search task becomes "ballistic": once started, it steers itself.

Stacking is not the only way that items can be serialized; unoccluded or "open" groups can be put into rows, for example, as in Figure 2.23(b). Of course, such arrangements are limited in their expressiveness, and richly expressive arrangements may violate seriality.

Contiguity. Iterating attention through a set is also easier when the items are *contiguous*, close together without interruption. All the forms of physical piles and files


(a) Serialized, occluded

(b) Serialized, "open"

(c) Unserialized; hard to guide attention.

Figure 2.23: Three pile structures varying in attentional demands

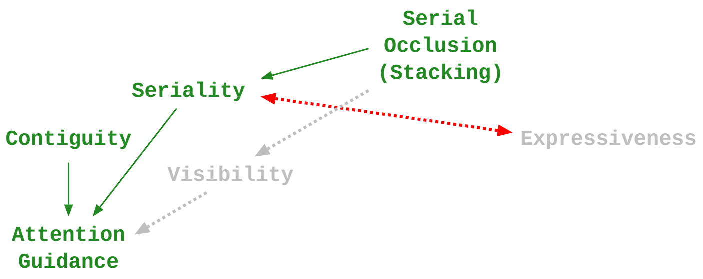


Figure 2.24: Stacking serializes piles. Serial structures, especially when contiguous, help to guide attention. However, seriality conflicts with expressiveness.

we've considered here have contiguity, but it's easy enough to imagine non-contiguous alternatives in a computer environment. In a computer workspace, meaningful groups of items can be distributed across space, possibly interleaved with items not belonging to the group. Such non-contiguous arrangements can be more expressive and flexible but may require more cognitive effort.

One technique for facilitating perception of non-contiguous sets is *highlighting*: members of the set are depicted with a unique visual attribute that "pops out", letting one's eye more easily travel the set. Highlighting is also used in *brushing*; see 3.4.2.

Aggregate Features. When the members of a set are visible, and especially when they are contiguous, the group acquires collective or *aggregate* (e.g. gestalt) properties. That is, the visual character of the whole can be qualitatively richer than the sum of its parts. For example, an unoccluded arrangement of items can have a *shape* that carries meaning or serves as a visual landmark in the "geography" of a collection [4].

Visual aggregation is reduced when items are occluded but persists even in a stack. For example, a stack can be *tall*, implying many members, or *ragged* (with misaligned edges), reflecting a history of intermittent interaction rather than a single episode of tidy construction. In fact, Apple's proposed design for digital piles [50] uses raggedness to distinguish user-built piles from computer-built ones.

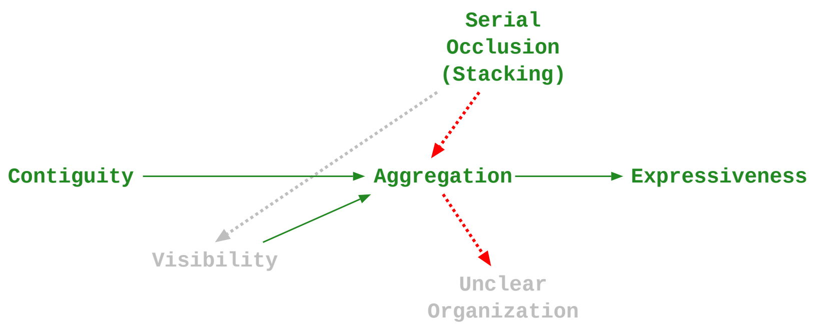


Figure 2.25: A pile acquires collective properties when its members are visible (and especially when contiguous) which make it visually distinctive. This can improve clarity and expressiveness.

By making piles more distinctive, and possibly more iconic, aggregation contributes to expressiveness and potentially improves clarity.

2.2.3 The System as a Cost Structure

The network of Figure 2.26 summarizes the influences we've considered in this chapter. It is one possible description of the cost structure of piling. The nodes of the graph represent several types of phenomena, including intrinsic qualities of individual piles (e.g. seriality, cohesion) and of the overall collection (e.g. smallness, expressiveness). But the majority represent some dynamic of a participant's *interaction* with the collection, such as the time course of organization (e.g. stability, flexibility) or the cost of various actions (e.g. accessibility, cognitive difficulty). In a loose sense, they're all affordances: each invites or deters certain interactions, and each interaction contributes to additional qualities, themselves affordances.

Chapter 1 sketched a cartoon version of this system: behavior in an environment is a negotiation between *what is easy* (its cost structure) and *what is needed* or *expected* (one's goals and prior practice). But even a relatively detailed and concrete representation like Figure 2.26 leaves no clean boundary between affordances, goals, and behaviors. The piler's strategies and goals, the physical structures and representations

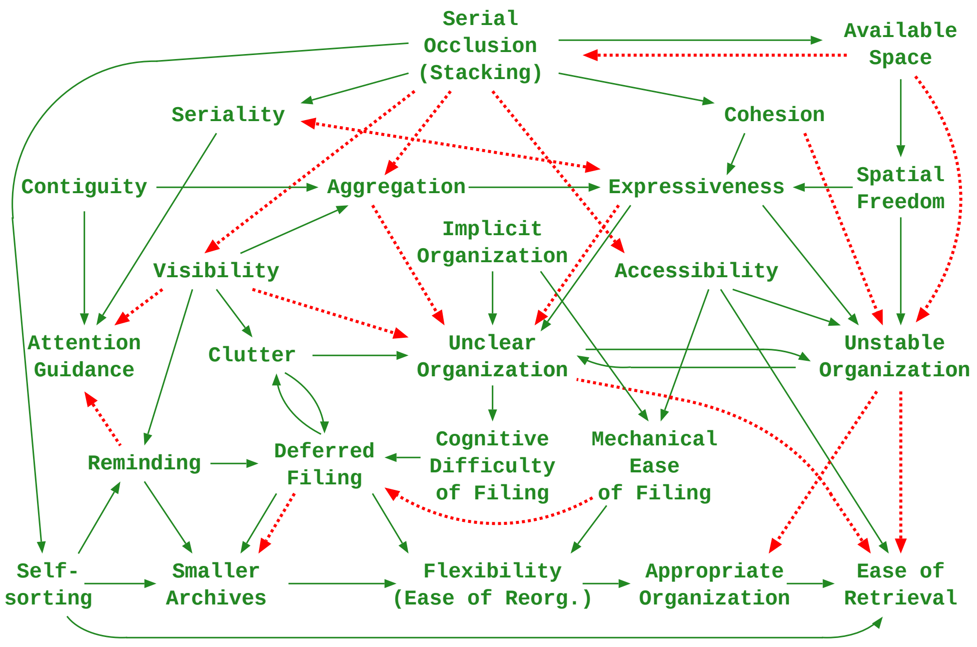


Figure 2.26: **The cognitive ecology of piling.** Summary of all interdependencies considered in this chapter.

they produce, and the qualities and affordances of those structures must be considered as a system.

2.3 Redesigning the Cost Structure

So far in this chapter, we've dissociated several aspects of piles and identified their contributions to the cognitive system. This description then helps us to *selectively engineer* the properties we want in a digital version of piles.

In this final section,we'll first introduce some related work that has applied various aspects of piling to digital domains — most notably, Apple's proposal for digital piles [50]. Then we'll assess Apple's piles in terms of the principles developed in this chapter.

That analysis has three purposes. First, it aims to validate these principles by applying them to an independent setting. Second, it will illustrate how digital environments can deliberately modify the cost structure of piling. Finally, that analysis will set the stage for Chapter 3, which describes how Dynapad alters that cost structure differently.

2.3.1 Related Work on Digital Piles

Characterized broadly, this research is about *personal information environments*, and that topic has been explored in others' work too numerous and varied to review here. Here let's consider briefly only a narrow subset of that work concerned with environments in which users emulate piling by organizing information manually by spatial (2D) position.

Data Mountain [56] and *Workscape* [48] are environments which are, like the simplest variant of Dynapad, primarily inert spaces which accommodate arrangements of document thumbnails.

Similarly, *MessyDesk* [21] lets users arrange text and images as a background collage on their system desktop. This essentially gives these items visibility without accessibility, as if they were under glass.

Henderson and Card's *Rooms* [32] environment addresses the problem of maintaining exposure in limited space. We might characterize their solution as *intelligent occlusion*: materials are organized into task-specific "rooms" so that exposure at any moment is allocated to the most relevant documents. We can also think of a room as an abstracted "fisheye" view [26] onto the activity structure of the document collection.

Dourish et al.'s *Presto* system and its *Vista* browser [19] focus on the flexibility of organizations. *Presto* offers "multivalent" documents which can appear in multiple places and dynamic piles which update their content automatically.

In addition to several environments for managing digital photo collections [57, 30, 8, 39], empirical work has compared various pile-like arrangements that users employ with photographs [31].

Apple's Piles

A seminal contribution to digital piles is from the Human Interface Group at Apple, as reported by Mander, Salomon, and Wong [50]. Mander et al. proposed several designs for a "pile" structure in the Macintosh desktop environment. Although these ideas are now nearly fifteen years old, they have yet to be integrated into the Macintosh (or any other) desktop environment.[6]

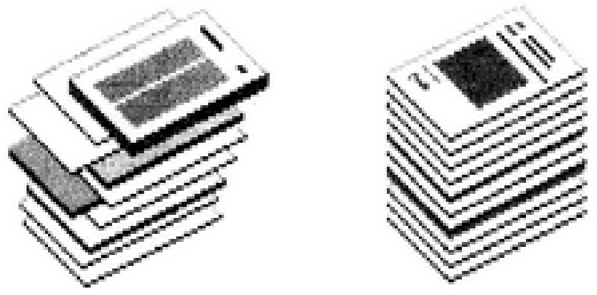


Figure 2.27: Apple's proposed piles: user-generated (ragged) and computer-generated (neat) (Images are taken from [50])

Apple's piles appear as axonometric-projected (3D-looking) stacks of document thumbnails (Figure 2.27). These stacks could be assembled either by hand (by dropping individual documents over each other) or by the computer (perhaps as the result of a search). To indicate this difference, user-built piles are left looking "ragged" and computed piles are aligned to look "neat". Although stacks have limited aggregation, this quality of raggedness is one such aggregate property, made more visible by the axonometric view.

The axonometric view has another advantage: it allows the user to index into the pile's depth using the mouse pointer, either to extract or to view individual documents within the stack.

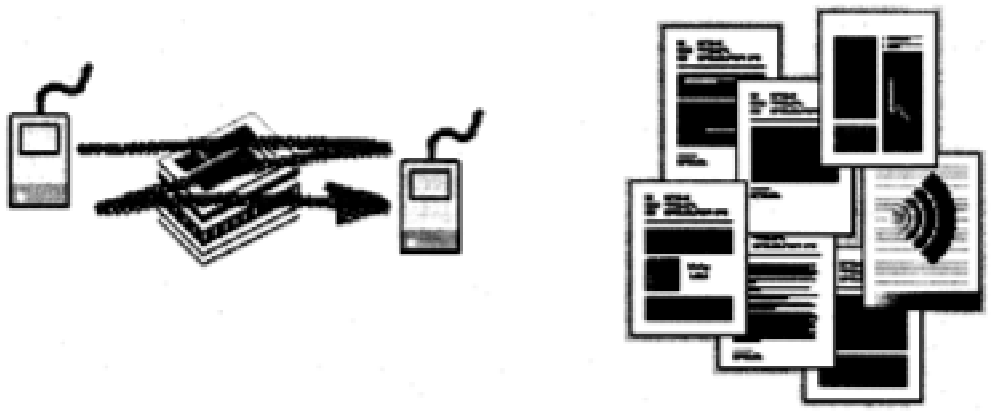
Design Variants. Apple compared several design variants differing primarily in two dimensions: whether the pile coheres together when stacked, and how the pile can be browsed or "opened".

Cohesion: Piles could be either "document-centric" or "pile-centric". With the document-centric version, dragging a piled item with the mouse would move just that item (possibly removing it from the pile). With the pile-centric variant, dragging a piled item would move the entire pile.

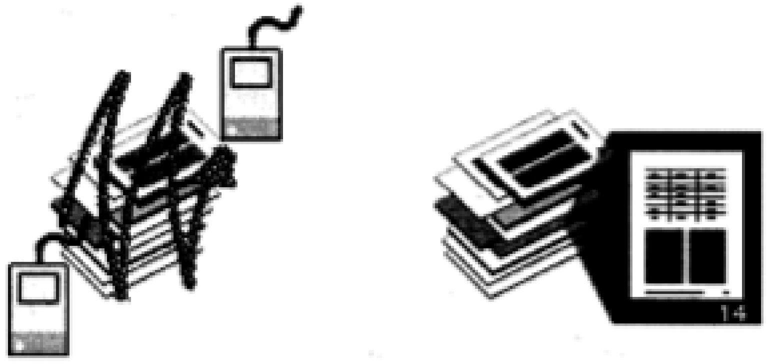
Mander et al. considered these choices to be exclusive and assumed that all piles

[6]Recently Saffer [59] partially implemented a prototype similar to Apple's design.

would behave the same way. We'll see later how Dynapad attempts to reconcile this whole-vs-part dilemma.



(a) "Scattering" a pile temporarily exposes all its members.



(b) Browsing a still-stacked pile with a "viewing cone".

Figure 2.28: Two ways of browsing Apple's piles (Images are taken from [50].)

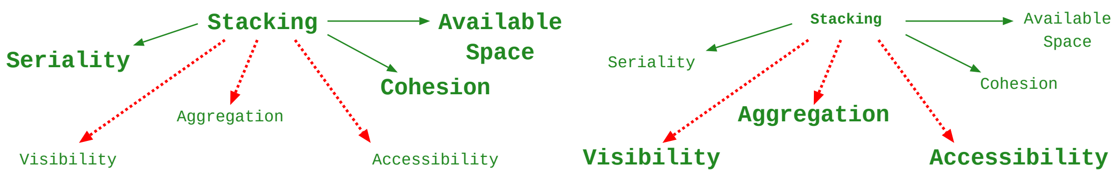
Browsing: Piles could be browsed in either of two ways. Rubbing sideways with the mouse would trigger *scattering* (Figure 2.28(a)), in which the pile contents would be temporarily unstacked and spread out to be visible and accessible all at once.

Rubbing vertically with the mouse would trigger a *viewing cone* (Figure 2.28(b)) which would display (and perhaps enlarge) one item at a time as the pointer was moved vertically through the stack. In this case, items would remain stacked; each could be seen but not all at once.

Apple also proposed several sub-variants of viewing cones differing in how the occluding items above the target are displaced. Additional proposals include turning occluders transparent [59]. For our purposes here, all of these variants can be considered equivalent.

2.3.2 Unstacking the Pile

We've seen that stacking entails tradeoffs; it improves some properties of piles at the cost of others. Specifically, stacking provides seriality, cohesion, and space, but limits aggregation and exposure (Figure 2.29(a)). Conversely, unstacking piles does the opposite, giving up seriality, cohesion, and space to improve aggregation and exposure (Figure 2.29(b)).



(a) Stacking (b) Unstacking

Figure 2.29: The effects of stacking vs. unstacking

This means that certain pairs of features are effectively in competition (e.g. accessibility and cohesion) but stacking can be used to mediate between them, to favor one or the other.

This interpretation helps us understand the design space of Apple's piles: each variation is a manifestation of that tradeoff. Let's reexamine each design variant in this light.

Pile-centric vs. Document-centric. The only reason for Apple to entertain both the pile- and document-centric variants, rather than combine them into a unified design, is that they represent a conflict.

One source of this conflict is an under-expressive input device. Specifically, direct-manipulation with a pointer (i.e. the mouse) has an ambiguous scope (i.e. one member vs. the entire pile). We can imagine the same problem trying to manipulate physical piles of paper with, say, a chopstick. But even then, we'd have several dimensions of control (e.g. angle, pressure, grip) that are unavailable in a mouse-driven interface. There are several ways of solving the resulting ambiguity, including ways of specifying more precisely the target of movement and other operations (see 3.2.2).

But there is another conflict at work here: when we depend on stacking for cohesion, cohesion competes against accessibility. Apple's pile-centric version offers cohesion, but poor accessibility. The document-centric version offers accessibility, the ability to manipulate documents individually, but no cohesion.

Splitting the design of piles into two separate variants attempts to address both the scoping ambiguity and this conflict. But that solution is unsatisfying: it resolves the ambiguities, but forces us to choose exclusively one affordance or the other.

Browsing by Scattering. Another solution is to trade-off cohesion and accessibility not into separate designs but into separate temporary modes — that is, to temporarily unstack piles.

"Scattering" the pile, as in Figure 2.28(a), is in effect a temporary unstacking, changing that pile's properties to those of Figure 2.29(b). The result is what we'd expect: increased visibility and accessibility, but at the cost of cohesion (the pile is no longer portable), available space (spreading takes up room), and seriality (items are disarrayed).

But let's consider more closely the last aspect, seriality. Although the scattered items in Apple's design are jumbled (perhaps to mirror the raggedness of the stacked pile?), they could just as easily be arranged into a serialized layout, such as a grid. That is, although stacking gives us certain properties for free, we don't always have to depend on stacking to get them. Often we can *synthesize* the desired properties (e.g. seriality) through an independent mechanism (e.g. regulated layout).

In doing so, we can eliminate conflicts between properties which are incompatible through stacking alone. In this example, we gain exposure by unstacking the pile, and then supplement its damaged "natural" seriality with a "synthetic" seriality. Figure 2.30 illustrates this manipulation.

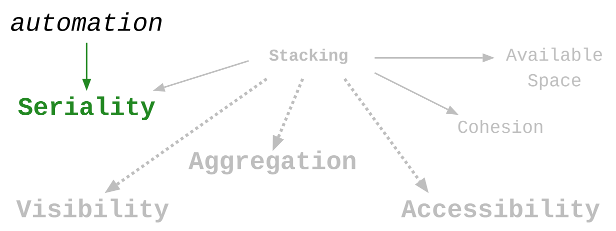


Figure 2.30: Regulated layout can supplement seriality

Browsing by Viewing Cone. Apple's other browsing mechanism, the viewing cone (Figure 2.28(b)), offers a similar manipulation: it synthesizes *visibility* in a pile which remains stacked. This lets visibility coexist with the benefits of stacking (seriality, cohesion, and space). To the extent that a user can also "act through" the viewing cone (for example, extracting a viewed document), this manipulation also augments accessibility, as in Figure 2.31.

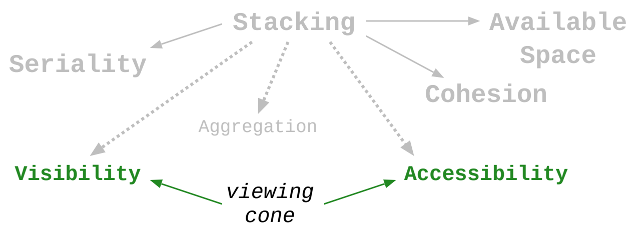


Figure 2.31: Viewing cones can supplement exposure

Note that neither visibility nor accessibility are as high as if the pile were unstacked: each document can be seen, but not without some effort both to initiate browsing and to locate the target. The cost of access or viewing is somewhere between that of a normally-occluded stack and a completely open "spread".

2.3.3 Looking Ahead: Dynapad's Manipulations

We've seen how Apple's design for digital piles manipulates the cost structure it inherits from paper piling to eliminate certain incompatibilities between desirable affordances. Dynapad employs this same basic strategy to manipulate the cost structure in additional ways. We'll see the details of Dynapad's design in Chapter 3, but now we can outline the basic approach.

Dynapad's approach is to unstack or "open" all piles and mitigate through other means the resulting damage to seriality, cohesion, and available space. Figure 2.32 summarizes these changes:

Seriality: Dynapad provides automated layout to serialize initially haphazard arrangements into grids and various linear formats.

Cohesion: Dynapad provides *containers* which protect the structure of exposed arrangements and let them travel as a group even while members remain individually accessible.

Space: Dynapad offers effectively infinite free space, made manageable by easily-controlled target-oriented zooming.

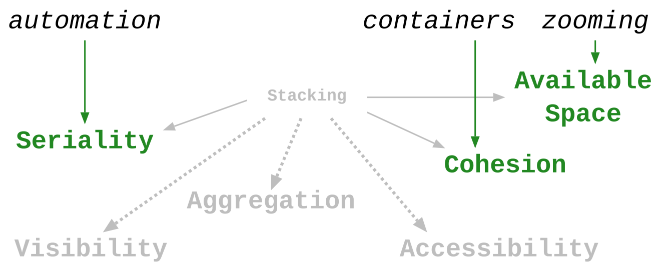


Figure 2.32: Dynapad supplements unstacked piles with features to improve seriality, cohesion, and space.

Indirect Effects. We must remember, of course, that these affordances occupy a position at the top of a watershed of influences, summarized in Figure 2.26. Every manipulation we make has the potential to change the cost structure downstream.

For example, consider the indirect effect of the changes above on *expressiveness.* Dynapad amplifies all of the properties which contribute to expressiveness (e.g. aggregation, cohesion), so we should expect expressiveness to improve also. But even before these manipulations, expressiveness has a tension with seriality, which we've magnified also. So now we can expect that tension to be even greater. How might we relieve it?

Dynapad's solution is to introduce *lenses* which allow alternative views of the same piles. One can be expressive while another is serialized. In this way, lenses help to *decouple* the mutual inhibition of seriality and expressiveness (Figure 2.33).

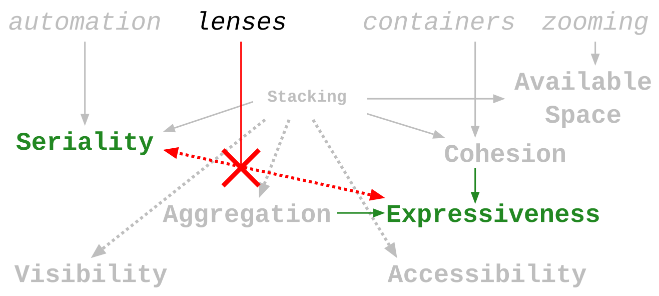


Figure 2.33: Dynapad also breaks the conflict between expressiveness and seriality by adding lenses.

Of course, Dynapad inherits other conflicts inherent in the system, as illustrated in Figure 2.26, and it introduces additional interactions and costs. Chapter 3 explores these features and trade-offs in agonizing detail.

3

The Design and Features of Dynapad

This chapter gives a detailed description of the functionality and interface of Dynapad's current design. Whenever possible, the description of each design choice includes its *rationale*, which may include several facets:

- historical precedents and development history;
- potential cognitive consequences;
- a characterization or *design space* of possible alternatives;
- abstract principles and constraints on that space;
- any conflicts, inconsistencies, or coordinations with other design aspects.

As with any cognitive environment, *details matter*. Although many of the details presented here are not critical to the overall functionality, they have implications for users' activity which will become apparent in Chapter 4. Therefore a second goal of this chapter is to provide a framework for discussing those details.

Dynapad's functionality can be abstracted into three levels:

Sections 3.1 - 3.5 present Dynapad's basic functionality as a largely inert virtual table-top. Digital collections can be arranged in a two-dimensional space just as papers are arranged on a table. The resulting activity has largely the same cognitive ecology as that described in Chapter 2.

Section 3.6 introduces *region-tools* which augment *space*. They offer a protocol by which the user can apply automation to certain areas of the workspace. (Much of this section has also been included in earlier work [3, 5].)

Section 3.7 introduces a *history* mechanism which augments *time*, allowing users to revisit earlier versions of a workspace and visualize the long-term evolution of their collection.

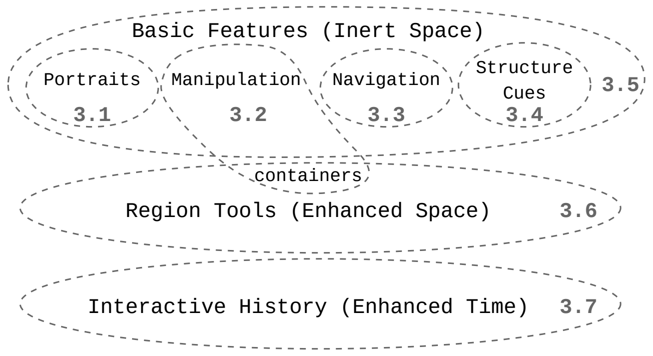

Basic Features (Inert Space)
Portraits
3.1
Manipulation
3.2
Navigation
3.3
Structure Cues
3.4
3.5
containers
Region Tools (Enhanced Space)
3.6
Interactive History (Enhanced Time)
3.7

Figure 3.1: Three levels of functionality and the sections of this chapter.

3.0.1 Overview of Architecture

My primary purpose in this chapter is to describe Dynapad's functionality and its rationale. However, some readers will benefit first from a quick overview of Dynapad's implementation and architecture.

Dynapad is currently implemented in two semi-autonomous layers. The first layer, written in C++, is the *renderer*, which is responsible for drawing the workspace and all objects in it. It uses the X11 [62] graphics libraries and therefore requires an X-Windows server running locally. The renderer controls the real-time display of zooming, panning, and animation, and it maintains a model of all graphics primitives which compose the objects in the workspace.

The renderer is a direct descendent of Pad++ [7], written by Bederson, Hollan, and colleagues. Many modifications to this version of Pad++ were contributed by Ron Stanonik and Ron Hightower.

Dynapad's second layer is the *physics manager*, which maintains a model of the collection's media elements and metadata and also controls Dynapad's interactivity and most of the features described in this chapter. It is primarily this layer which represents Dynapad's "design" as considered in this document and which dictates the affordances guiding users' behavior. In order to explore the design space more easily, we have implemented this layer in PLT Scheme [24], a lightweight object-oriented language convenient for prototyping. I have been the primary developer of this layer, and it is in that respect that I refer to myself as "the designer". However, Dynapad is a deeply collaborative effort, in both design and implementation, and owes much to the inseparable contributions of Hollan, Stanonik, Hightower, and others.[1]

With that glimpse into Dynapad's implementation, let us turn now to its functionality.

3.1 Collection Elements

Our research emphasizes the value of *visual* access to information. So a natural medium for Dynapad is one which is already visual: digital photographs. Like many other applications for browsing digital photo collections [57, 30, 39, 8], Dynapad displays photos as thumbnails which can be arranged and "piled" around the workspace, as we saw earlier in Figure 1.1. Section 3.5 describes some of our preliminary studies of users organizing their photo collections in Dynapad.

3.1.1 Document Portraits

However, we intend for Dynapad to support many other types of content, which must then be converted into a graphical form. So far, Dynapad supports not only images but PDF documents. Because images from these papers can be effective retrieval cues, Dynapad extracts the component images and collages them into "portraits" or "enriched thumbnails" [71] of the documents. Figure 3.2 shows several sample portraits.

[1]See page xiii for specific acknowledgments.

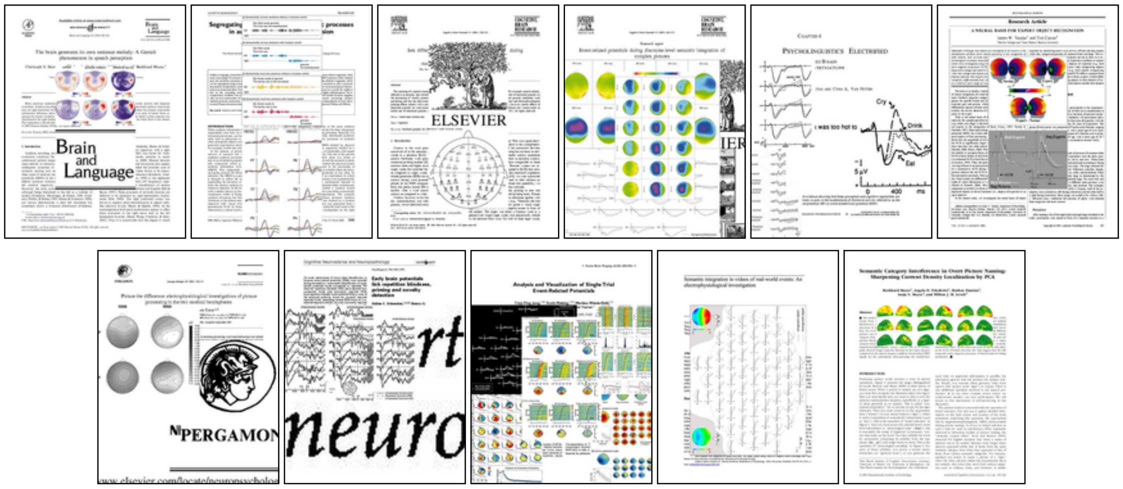


Figure 3.2: **Sample PDF document portraits.** These are Dynapad's default portraits for a sample of "ERP-related" documents belonging to subject **B**.

The value of such portraits can be illustrated with an example[2] from our observational data. The portraits in Figure 3.2 are of documents belonging to **B**, one of our research subjects. **B** selected this particular set when looking for "ERP[3]-related" research papers in her collection. In her view at that time, these portraits were only about 92 pixels high, about the same size shown here. Yet from these representations, **B** was able to recognize a very complex category of content, based on visual features evocative of that genre (e.g. clusters of waveforms and brightly-colored brain "maps").

Generating and Editing Portraits

Currently, the algorithm Dynapad uses to generate document portraits applies very simple heuristics:

1. Extract all embedded images and sort them by file size. This reflects both the size and complexity of an image, and therefore its probable information value.

2. Discard any images smaller than 5KB or with aspect ratios more extreme than 5-to-1 (which are usually background textures).

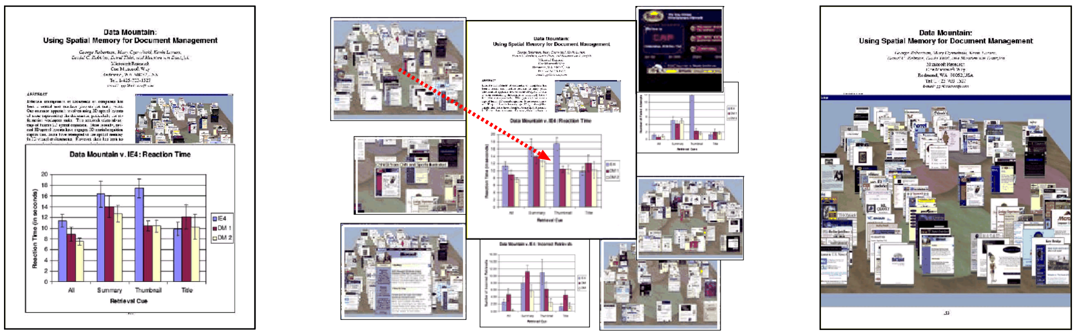
[2]For details of this episode, see page 216.

[3]ERP ("Event Related Potentials") refers to a cognitive neuroscience methodology for analyzing brain activity from electrodes on the scalp.

3. Rescale and arrange the top few images in a pre-set pattern (e.g. one alone is centered, two are stacked, etc.) over a background image of the document's cover page. This usually leaves visible its title, authors, and basic typographic style, which often suggests "family resemblances" to other documents from the same source.

Clearly, much more sophisticated techniques could be applied, including the judicious enlargement of important text like keywords (as with [71], for example).

But of course, no algorithm can always guess correctly what will be the most effective portrait for a given paper. Therefore, after making its best guess at a default portrait, Dynapad lets the user edit any portrait, to emphasize the elements she finds most informative. The process of "unlocking" and editing a portrait is demonstrated in Figure 3.3.


Data Mountain:
Using Spatial Memory for Document Management
Data Mountain v. IE4: Reaction Time
Reaction Time (in seconds)
Retrieval Cue
All
Summary
Thumbnail
Title
IE4
DM 1
DM 2

(a) An initial locked portrait.

(b) Portrait is unlocked, allowing rearrangement of its images and additional hidden images.

(c) New arrangement is cropped and locked.

Figure 3.3: Editing a PDF portrait of [56]

When a portrait is unlocked, all component images (and any hidden images) may be moved and resized (although they are forced to stay contiguous with the background image). When locked again, the collage is cropped to the boundaries of the background and unused images are "stored" out of sight and may be accessed later if the collage is again unlocked.

Interactivity

Both photos and portraits (which are themselves images) display as low-resolution thumbnails most of the time, since they are usually too small to show details anyway. However, they automatically substitute a high-resolution version when the user zooms into them.

Two zooming events will trigger the high-resolution version: double-clicking an image, which zooms in automatically, or "pulling" it manually closer until it stops, as described in section 3.3.1.

Browsing Documents. Additionally, portraits and images can be made to launch "helper" applications when clicked (for example, an image-editor for photos). Currently, Dynapad launches a PDF-browser application when a portrait is double-clicked with the second (navigation) mouse button, or through a pop-up menu. The browsing application opens the PDF document corresponding to that portrait.

While this feature has proven very useful, we have noticed a cognitive hazard with it. Because there is no visual continuity between Dynapad and the browsing application, once the browser is closed, the user may have trouble re-locating in Dynapad the portrait that triggered it. A solution may be to zoom automatically to a browsed portrait, or else to select or highlight it uniquely.

3.2 Interacting with Objects: Selection and Movement

Now that we've considered the basic properties of a workspace's primary contents (photos and portraits) let's turn to the basic physics of interaction: navigating around the workspace and moving those objects within it.

We might regard this as the "whitespace" of activity: gripping and moving objects and redirecting our attention is a fluency we take for granted in the physical world, not a salient or glamorous "feature" of the environment. But that whitespace consumes a sizeable fraction of the overall effort of interaction, and it must be as carefully designed as any explicit feature. Details matter.

Dynapad is by no means the first to address these problems; indeed, one of the design constraints is to be consistent with the expectations established by countless other applications. But to understand the impact of these details on the overall ecology, it's helpful to return to first principles and examine *why* so many direct-manipulation interfaces are designed as they are.

3.2.1 The Poverty of Input

In a natural ecology, one of the principles useful for understanding complex interactions is that of *competition for limited resources.* A similar constraint is present in a software interface: any input device has only a small number of degrees of freedom. For example, a typical three-button mouse (without a scroll-wheel) has five independent variables: two continuous (X and Y position) and three binary (the buttons). These define the device's *input space* (also called its "movement vocabulary" [15]).

Input vocabulary is a scare resource. The number of operations needed to interact richly with a world far exceed the number of independent input channels. Therefore each input action must be *overloaded*, assigned multiple meanings in different contexts. Context can vary in many ways, including *location* (e.g. clicking a button vs. an object), *timing* (e.g. double-clicking or gesture strokes), and *modes* (i.e. earlier inputs establish special "states"). Dynapad's interface must be understood as a product of this competition and overloading.

Alternative Input Devices. Dynapad's default interface is designed for the three-button mouse, and the descriptions throughout this chapter will focus primarily on this configuration. Nevertheless, we anticipate the need to adapt the interface to other devices which may have different (and often fewer) input channels, and occasionally the interface description will include these variants.

Most obvious is the two-button mouse of Windows and Linux systems. The third button is a luxury but not necessary; we intend Dynapad to work unimpaired with only two. One button, however, will be difficult, as we shall see.

The DiamondTouch Table. Additionally, we have begun to adapt Dynapad to work with various tabletop devices, most notably the DiamondTouch table [17]. This is a touch-sensitive horizontal surface which replaces the mouse by sensing contact from a user's bare hands. The computer's display is projected onto the surface and users interact directly with that space.

Although this style of interaction has several advantages, the table's current version is designed to emulate only a one-button mouse.[4] We adapt it to essentially two buttons by "stealing" a binary value from the edge of the continuous position variable. See section 3.3.4 for details.

Modifier Keys. The scarcity of input variables on a mouse or table can be relieved without overloading by using modifier keys (like *Shift* or *Control*) on the keyboard. But this requires two hands and is inconvenient, especially when using a table input. In our experience, it increases the cost of those actions enough to deter their use if there is a one-handed alternative. Dynapad's goal is to provide keyboard *shortcuts* but require only a mouse (or table) for all elemental actions.

3.2.2 Operational Syntax and Selection

Another theme is the need for *operational syntax*, the conventions for how operations are applied to objects. For example: in most text editors, copying text requires two actions, first selecting the text fragment, then specifying a copy operation, via menu or keystroke. The operation (what to do) and object (where to do it) are specified in two separate actions, the object first in this case. This example has a "postfix" or "serial object-verb" syntax.

The reverse syntax, "prefix" or "serial verb-object", is also common. For example, the tool palettes in many graphics applications require selecting first a tool (e.g. "fill") and then a location to apply it.

Operational syntax need not be serial. Consider the ToolGlass [12] interface: one hand moves a semi-transparent virtual palette, and the other controls a pointer. An action and object are specified at the same time by superimposing a particular tool on the palette over an object, then "clicking through" with the pointer. That tool (operation) is applied to that object with a single action. Similarly, a ToolStone [55] is a physical

[4]Actually, DiamondTouch only emulates a *half*-button mouse: its two continuous dimensions (position) only have a value when its single binary dimension (contact) is "on".

input device with multiple faces, one for each of its operations, which is oriented and touched to objects or locations to apply that operation.

Note that the possibilities for operational syntax are limited by the poverty of input. For example, the two-handed parallel syntax of ToolGlass requires four independent continuous input variables (X and Y in each hand) and is possible only with a second mouse or a more complex input device. And the ToolStone, although one-handed, requires at least one channel beyond position to specify its orientation. Without these extra input channels, syntax must be serialized.

N-ary operations. This discussion has only considered unary operations, actions on only one object (or a set of objects which are all treated the same way). Even more overloading is required for N-ary operations, which involve multiple objects in asymmetric roles (for example: "make A the same color as B"; "space A and B like C and D"). Dynapad currently has only unary operations, in part to avoid those complexities.

Operation Scope and Selectors. A second simplification so far has been considering the object of a operation to be specified by a single point. This is usually adequate when objects are discrete and spatially separated. But there are cases when a single point may not be enough the specify the full scope of the operation:

- Indistinct objects and regions of space;
- Multiple objects;
- Overlapping and hierarchical objects.

The common solution is to use a *selector*, which serializes the single input point to specify a more complex scope. For contiguous regions and uninterrupted sets of objects, a selector can simply specify the spatial boundary defining the region or containing all the objects. The two most common versions of this are the *lasso* and the *box*. In the discussion ahead, "selector" will refer to both of these variants.

But even these are inadequate for non-contiguous regions or diffuse sets of objects. To solve this, selectors may be made *incremental*, able to add to or subtract from a prior selection. This usually requires further serialization of the syntax.

Prefix vs Postfix. The need for serialized selectors gives an advantage to postfix (object/scope first) syntax over prefix (operation first) syntax. A prefix operation with a potentially complex scope requires a "go" action after the selection process to initiate the pre-specified operation. But with postfix syntax, choosing the operation itself marks the end of selecting, and the operation then applies to whatever scope is selected. For this reason Dynapad, like most programs, uses (mostly) postfix syntax.

Reusing partial syntax. Successive operations can be simplified by reusing part of the specification of earlier operations. With prefix syntax, for example, it saves effort to default to the last operation and merely reapply it to a new scope. This is how tool palettes typically work; choosing a tool specifies a prefix operator which applies to all targets until a new tool is chosen. This can also be a way around the problem of complex-scoped prefix operations: instead of specifying the entire scope incrementally and applying the operation (e.g. "paint all these together"), apply the operation incrementally to subsets of the eventual scope ("paint these, then those").

Conversely, successive postfix operations can share the same scope if they each leave those items selected. Like most applications, Dynapad preserves selections until scope is reset (more on this shortly).

Finally, it can be possible to transfer relative scope: that is, transfer a complex scope to different objects with an "analogous" configuration. Dynapad's *region tools* afford such transferable scope through portable boundaries. See section 3.6 for extensive discussion.

Selection for Cohesion. The most important use of arbitrarily-complex selections is to emulate *cohesion* of groups, as seen in 2.2.2. In the basic Dynapad application, a "pile" is (usually) an open cluster of adjacent items, which may be invested with internal structure. The ability to select the pile and move it as a group preserves that structure and helps the stability of the collection's organization.

With paper, cohesion requires stacking and therefore occlusion. But in a digital environment, multi-item selection breaks the coupling between occlusion and cohesion: we can leave piles open and still move them without disruption. Later (3.2.4 and 3.6) we'll see another way to keep piles coherent, by binding them together into explicit groupings or “containers”.

Furthermore, incremental selection (discussed next) allows selections which are non-contiguous. So selection also decouples cohesion from contiguity.

Incrementing and Inverting a Selection. By default, a selector clears any existing selection and starts anew. But like many applications, Dynapad provides an *incremental* selector which doubles as an *inverter*. Holding the *Shift* key while starting a selector (“shift-dragging”) makes it an inverter: whatever is enclosed is *added* to the selection or *removed* if already selected. Additionally, “shift-clicking” (or “shift-tapping”) toggles (inverts) just the immediate target.

This Shift-plus-button combination is a well-established standard, but we should remember that it is equivalent to any other binary input channel. A different modifier key could be used, or the combination replaced by a separate (perhaps the third) mouse button, which consumes a precious input. Because incremental selection is convenient but non-essential, we’ve chosen to use a modifier key and save a mouse button for other operations. But any reference to Shift-click or Shift-drag should be interpreted as an abstract action that can be reassigned to other inputs.

The entire selection is cleared simply by clicking the background to select nothing. Shift-clicking the background changes nothing.

Direct movement: bypassing selection. Although a general selection mechanism is needed for certain operations, there is tremendous value in the ability to manipulate objects directly [67], without needing to think about syntax or selecting. This is especially important for the most common operation: moving objects in the workspace.

One of Dynapad’s sacred rules is always to allow direct-manipulation movement of items, regardless of what is selected. Any portable item can be moved simply by clicking and dragging on it at any time (with a few difficult exceptions, explained later). Movement is a special operation whose scope is determined by this rule: *if the immediate target is selected, move the entire selection; otherwise, move just the target.*

Resetting the selection. In the first case above, if the entire selection is moved, there is no reason to change the selection. But the question remains, in the second case, what to carry forward as the default selection. There are four obvious possibilities:

1. keep the current selection, ignoring the target;
2. add the target to the selection;
3. clear the selection completely;
4. clear the selection then reselect only the target;

As always, there are trade-offs. One helpful heuristic in comparing these options is to *keep actions atomic*: don't combine dissociable effects automatically if their combination can be emulated easily by doing each separately. By this rule, options 1 and 3 are better than 2 and 4, since the target can easily be added incrementally after it's moved in either case. Without knowing the user's intent, it's safer to automate too little than too much.

This rule also suggests that 1 is better than 3, since 1 changes less. It preserves any investment in a complex selection but lets the user emulate 3 by clearing the selection manually.

However, a little experience with this implementation reveals a chronic hazard: after moving a few unselected items, one's attention has moved away and often forgotten the persistent selection, and future operations can have unwanted effects.

For this reason, Dynapad currently uses option 3: moving an unselected object clears any selection, and nothing needs to be remembered. If a complex selection is mistakenly de-selected, it can be re-selected with the *undo* action (see 3.7).

Overloading Movement and Selection. As two of the most frequent actions, moving and selecting should both be as easy as possible; certainly they both merit a mouse button. But fortunately, they can share a button. We overload mouse button 1 to control both movement and selection, and disambiguate by the starting position: starting to drag on an object moves it, and starting on the background creates a selector.

3.2.3 Selection boxes and lassos: a micro-ecology

So far we've lumped together the two common forms of selectors, the box and lasso. Both are defined by using a mouse (or other input device) to drag a point-cursor along a path.A selection box uses the start and endpoints of the path as opposite corners of a rectangle, and selects all objects enclosed by that rectangle. A lasso uses the path itself as the perimeter of a loop, automatically closed by connecting the start and endpoint, and selects all objects enclosed by that loop.

These alternatives offer different advantages. A box requires less dragging (since diagonals are shorter than perimeters) and is more robust to sloppy dragging, since it ignores the path between the endpoints. However, it may require considerable precision at the start and end. More specifically, starting and ending a box requires one to scan visually both horizontally and vertically along the edges of the box to ensure that they enclose only the intended targets (Figure 3.4). Starting requires additional effort, since the edges, not yet made, must be imagined. It also may require thinking "around the corner", anticipating where the closing edges will fall. Rather than imagining the final solution, users may converge on a solution in several tries. The early tries serve as *epistemic actions* [44], saving cognitive effort by probing and reacting instead of simulating the process mentally.

In contrast, the lasso requires a much simpler perceptual task throughout the dragged path: simply ensure that path goes to the correct side of the nearest object.

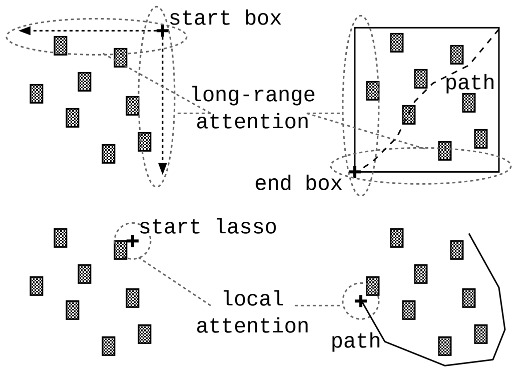


Figure 3.4: A selection box requires long-range attention; a lasso localizes attention.

Put another way: both selectors require some cognitive effort to use, but they differ, as cognitive artifacts, in how they mediate that effort. The lasso distributes and guides the effort, while the box concentrates it all into two relatively complex decisions, at start and end.

The effort required for each tool also depends on the configuration and relative density of the targets and their surroundings. If the targets are in a roughly rectangular arrangement (such as a collection's initial layout grid) or in a cluster well-separated from non-targets, a box may be easiest; otherwise, a lasso may be better. The cost structure of using either tool depends on how objects are generally arranged, which will evolve during interaction.

Finally, boxes tend to be more effective when the view is zoomed out and objects are small: their indifference to path pays off more and their long-range attention costs less, since all distances are reduced. On the other hand, they may also beat lassos when zoomed in close, which magnifies the lasso's cost of traveling a long perimeter instead of a shorter diagonal.

Containment Rules. There is another subtle but important consideration: what exactly does it mean for the selector to "enclose" an object? What containment rule should be used? Four obvious candidates are:

any: An object is selected if any part of it lies within the border.

all: An object is selected if it lies completely within the border.

most: An object is selected if more than half of its area is within the border.

center: An object is selected if its center point is within the border. The definition of "center" is flexible (e.g. center of bounding box vs. "visual" centroid).

For symmetrically-shaped (e.g. rectangular) objects, the **most** rule is effectively equivalent to the **center** rule, so let's disregard **most**. What are the consequences of each of the remaining three rules?

At first glance, both **any** and **all** seem to have an advantage of predictability. Assuming the edges and corners of objects are visibly sharp, it's easy to see where the selector border intersects an edge but harder to guess where the center is.

Consider Figure 3.5: the intended selection should include the objects on the upper left. When we imagine the "ideal" border isolating those, it falls in a visually salient zone, a zig-zag strip between objects. The easiest action is to approximate that ideal line with a straight line that cuts corners. With either the Any or All rule, that

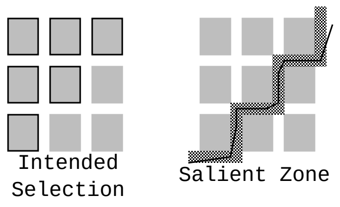


Figure 3.5: When selecting the upper left items, the most salient path follows the zig-zag margin.

will be an error, including too many or too few objects. But the Center rule includes only the intended objects.

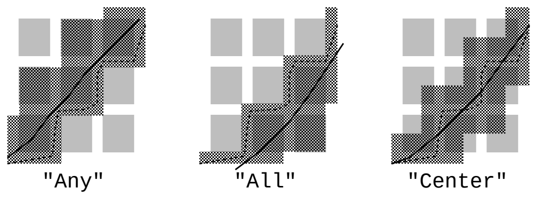


Figure 3.6: When taking a diagonal shortcut, the **center** rule is most stable.

We can think of it in terms of stability. For each of the rules, there is a "safe zone" in which any path will select the intended objects. The wider a zone is, the more stable it is against the sloppiness of drawing. The narrower it is, the more sensitive it is to imprecise dragging. In this example, all zones are equally wide and stable. But the **any** and **all** zones are misaligned with the Salient region, and therefore counter-intuitive. They are also asymmetric; we must remember which rule is used and which side of the line is "inside". If we forget, the "safe" zone is the intersection of the two (the salient margin zone), which is much more narrow and unstable. The safe zone for the **center** rule is stable, symmetric, and salient at the same time, and so requires less effort and precision to use without error.

Discriminability. We've seen that the **center** containment rule is generally best, especially for a lasso, which benefits from cutting corners and straightening lines. But there are cases where the *discriminability* of the other rules is an advantage. That is, **any** and **all** can dissociate items which are very close together or slightly misaligned. Any portion of an object which extends beyond the edge of others can be used as a "handle" to isolate it with a discriminating selector, as in Figure 3.7.

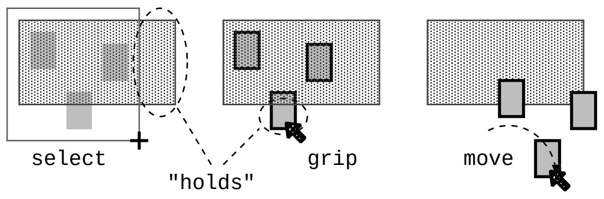


Figure 3.7: Selecting some overlapped objects using the **all** rule

So if the **center** rule is better when cutting corners, but **any** and **all** are better for selectively gripping nearby items, can we make both advantages available? Yes: by assigning different containment rules to the two selectors. Dynapad uses the **center** rule for lassos and the **all** rule for boxes.

The choice of tool can be used to leverage the variability in attention mediation, diagonal stability, and discriminability, whichever is most important for a particular situation. Neither tool is universally better; their relative costs and the conditions in which one is superior change constantly throughout interaction. To use them well, a user must implicitly understand all these trade-offs. But when both tools are available, the user may spend additional cognitive effort monitoring their success with one and switching to the other. And if the cost of switching is too high, they may simply stick with the current tool through conditions when it is ineffective.

Integrated Lasso and Box. With these costs in mind, Dynapad integrates both box and lasso into a single selection mechanism (inspired by a similar feature in [61]).

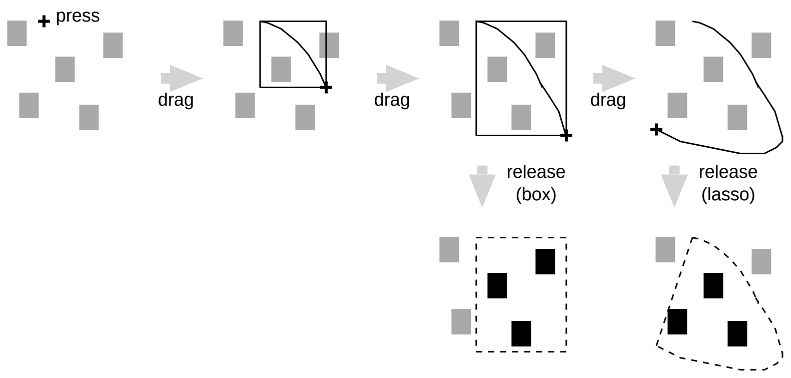


Figure 3.8: Dynapad's integrated selector. The box is used until the path curves into a lasso.

Both variants share the basic action of dragging a path, but a lasso's path must eventually bend significantly, while a box's diagonal remains roughly straight. Dynapad simply maintains both tools, sharing the same path, until the ambiguity is resolved. If the drag is diagonal and ends without curving, the selector uses the box; otherwise it uses the lasso[5]. If treated like a box, it behaves as a box; if treated like a lasso, it becomes a lasso. Absolutely no additional effort is required to switch from one interpretation to the other, letting the user react more fluidly to the particulars of each situation.

3.2.4 Containers and Dragging

In section 3.6, we'll explore various *region-tools* which extend Dynapad's functionality beyond the basic "inert desktop" physics. But here let's preview one class of these tools, *containers*, which complicate the overloading of selecting and dragging.

A container behaves like a portable tray which, when moved, carries any objects (portraits or other trays) placed on it. It embodies a hybrid metaphor as both an object itself and a space, like the background, in which other objects are selected and moved. But since dragging is overloaded for both moving and selecting, dragging a container creates a dilemma: as an object, it should move, but as a space, it should begin a selector for its contents.

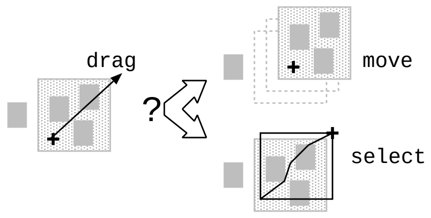


Figure 3.9: Should dragging on a container move or select?

One solution to this dilemma is to divide a container into separate zones: a "solid" handle for moving it (like the title bar of a window) and a "hollow" region in which to arrange and select members (like the body of a window).[6] Dynapad avoids this for two reasons. First, many containers are not rectangular (see 3.6.1) and their irregular

[5]Currently the actual decision rule is simple: use the lasso if the box ever narrows to less than half of its maximum height or width.

[6]In some cases, a third zone (the border) is needed for resizing.

shapes are difficult to subdivide consistently and intuitively. Second, because in the multiscale workspace objects vary dramatically in size, it's difficult to maintain a handle which remains large enough to use but doesn't interfere with the container's contents.[7] Therefore a container's body is homogeneous; dragging anywhere in its interior (but outside any members) has the same result.

That result (moving or selecting) varies between two generations of implementation, described below (3.2.4–3.2.4). Broadly speaking, normal dragging moves a container, and shift-dragging selects (inverts) items within it.

Layering and Scoped Selection. Containers introduce different *levels* of space. That is, the immediate members of a container are at one level, but the container itself is one level "down". All "loose" items on the workspace surface, including the outermost containers, are at "ground" level.

One advantage of dissociating these levels is that a container can safely overlap ground-level objects without disturbing them or its own members, as in Figure 3.10. In this way, containers can afford occlusion[8] to entire groups, with the advantages considered in Chapter 2: space can be reclaimed, and the occluded structures can be intentionally hidden, perhaps to reduce distraction.

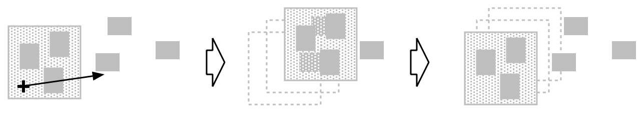


Figure 3.10: Covering ground-level items with a container

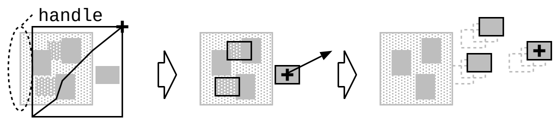
Originally we assumed a perimeter-selector to be sufficient in a two-dimensional workspace. But layering of objects makes Dynapad essentially "two-and-a-half" dimensional [7]. So layering forces another design choice about selectors: should they include all levels of items within their perimeter, or use levels to filter the selection? Dynapad chooses the latter: the starting point of a selector (or inverter) determines its level, and it includes only items in that level. The default selector can enclose only ground-level items (including containers), and a shift-selector within a container can enclose only that

[7]A similar problem exists for readable labels; see 3.4.3.

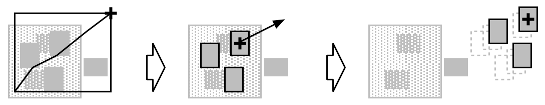
[8]Currently, only the first stacked layer is reliable. Stacking containers on each other often *nests* them instead.

container's members.

This constraint makes it easier to separate layered items. Items in the top-most container can be shift-selected as a continuous block without mistakenly including occluded items below, or the occluded layer can be extracted from underneath the container (Figure 3.11). The latter trick is aided by a box-selector's **all** containment rule, which can exclude the container itself if any portion (a "handle") extends beyond the desired selection.



(a) Selecting and extracting at ground-level



(b) Shift-selecting and extracting from container

Figure 3.11: Selecting from layered items

Overloading *Drag*, with Containers (First Generation)

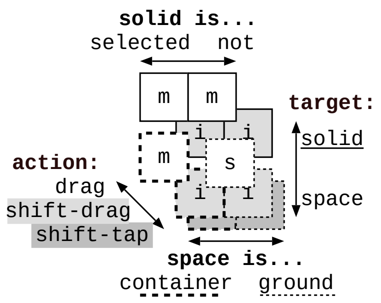
Two different implementations of Dynapad resolve differently the ambiguity of dragging containers. The simpler version treats a container like an object (moving it) for normal dragging, but like a space (selecting within it) for *shift*-dragging. Shift-dragging an inverter at any level affects only items in that scope, i.e at ground-level or within the container.

Summarizing all the considerations so far: all forms of selection and movement are accomplished with a single mouse-button (plus the *shift* modifier-key). Dragging with that button is overloaded with ten different conditions, determined by five binary parameters:

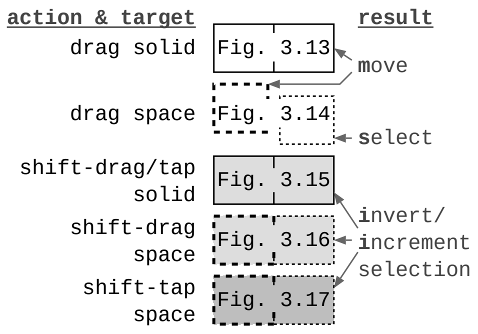
- Is the action a normal or *shift*-drag?
- Is the action a long drag or a short tap (no motion)?

- Is the target a "solid" object or a potential space (container or background)?
- If a solid, is it already selected?
- If a space, is it a container or the background?

Figure 3.12 summarizes these cases and organizes them into five pairs, illustrated in Figures 3.13–3.17. All ten examples share the same initial condition and vary only in the drag action and target.

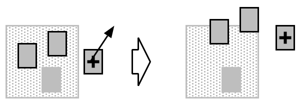


(a) Five binary parameters determine one of ten possible conditions.

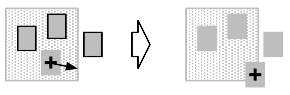


(b) Those conditions, detailed in Figs. 3.13–3.17, result in either moving or selecting.

Figure 3.12: Overloading the meaning of a *drag* input



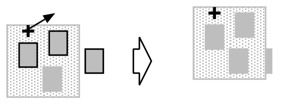
(a) *Target selected:* Move all selected



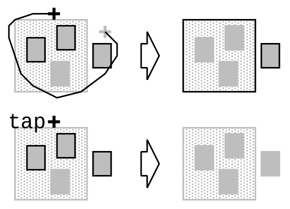
(b) *Target not selected:* Unselect all, move target

Figure 3.13: Dragging or tapping on a solid

Violations of Metaphor. The various conditions of dragging suggest two incompatible metaphors: *space-like* dragging which encircles objects with a selector or inverter, and *solid-like* moving or toggling of items. Empty space is consistently space-like, and portraits are consistently solid-like. But the inevitable compromise with containers creates three apparent inconsistencies (illustrated in Figure 3.18):

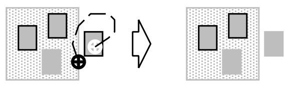


(a) *On container:* Unselect all, move container (behaves as solid)

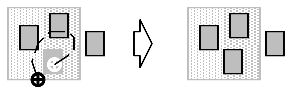


(b) *On ground:* Unselect all, reselect only on ground. Tap selects nothing.

Figure 3.14: Dragging or tapping on space

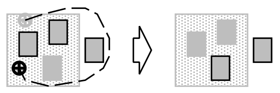


(a) *Already selected:* Invert object

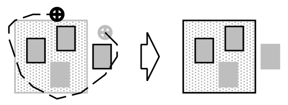


(b) *Unselected:* Invert object

Figure 3.15: Shift-dragging or tapping a solid

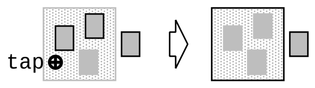


(a) *On container:* Invert only within container

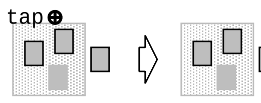


(b) *On ground:* Invert only on ground level

Figure 3.16: Shift-dragging space



(a) *On container:* Invert container itself



(b) *On ground:* No change (equals empty inverter)

Figure 3.17: Shift-tapping on space

1. Dragging at ground-level is space-like, but dragging in a higher level "space" (a container) is solid-like. This violates a reasonable expectation of parallel behavior at all spatial scales. This expectation and resulting violation is especially strong

when the container is effectively invisible, a closeup view in which its edges are off-screen and only its color and texture provide a visual clue. Treating it as ground and trying to drag a selector instead moves that apparent ground – which looks exactly like *panning* the view, accomplished with a different mouse button (see 3.3.2). This was one of the most frequent errors made by our observed participants (see 4.2).

2. Dragging a container is solid-like, but *shift*-dragging it is space-like. For normal space, the meaning of *shift* is *inversion*, but within a container it also changes *solidity*, a completely independent concept.

3. Shift-dragging a container is space-like (inverting members), but shift-*tapping* it is solid-like (inverting the container itself). The difference between these actions is only how far the mouse travels with the button down; a small quantitative change produces a discontinuous qualitative change in behavior.

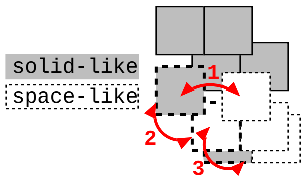


Figure 3.18: Containers' hybrid behavior, both solid-like and space-like, entails inconsistencies (1-3).

These inconsistencies could be eliminated by dissociating space-like dragging (selecting) from solid-like dragging (moving), by assigning them to different inputs: either another modifier key (which inconveniences one of the ubiquitous actions) or separate mouse buttons (which are prohibitively scarce). All of the complications drag-overloading and mixed-metaphors can be traced back to the poverty of input.

Multi-Phase Containers (Second Generation)

It would be desirable to give containers either a consistently solid-like or space-like behavior. But either choice creates conditions where needed functionality is missing. If containers are consistently space-like, they cannot be moved by dragging. If they're consistently solid-like, they members cannot be easily accessed. Note that this is exactly the same conflict as that behind Apple's pile-centric and document-centric design variants: a tradeoff, once piles are unstacked, between accessibility and cohesion.

Fortunately, these functions are needed at different viewing scales: cohesive moving is least important at very close zooms, and manipulating individual members is least important at very distant zooms, when items are too small to see anyway. So we can vary containers' behavior with zooming, choosing at each scale the single metaphor whose absent functionality is least likely to be missed.

Freezing. A container's space-like role concerns the arrangement of its members.At very small viewing scales (zoomed out), the display resolution becomes too low to see or manipulate these arrangements safely. A container's contents may be so small and close together that a selector cannot reliably separate them, eliminating the need for space-like dragging.

Additionally, it becomes difficult to grip and move the container without instead grabbing one of its members, which may be difficult to replace precisely. This not only disrupts the container's internal organization (instability!), it provokes additional frustrating attempts to grip and move it. In short, the container is too small to play any role as a space.

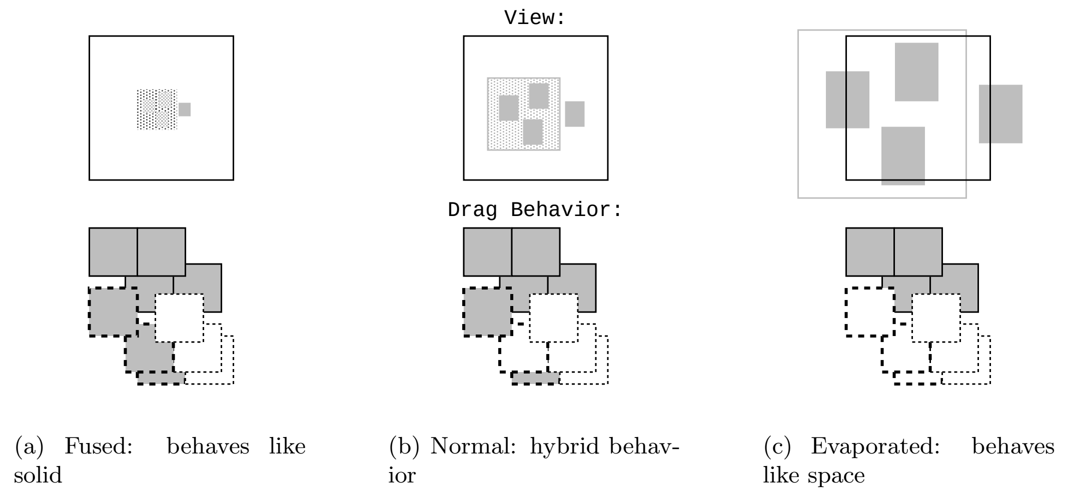
Therefore, containers below a certain size[9] *freeze* or *fuse* together to behave consistently as a solid. Their usual space-like action, shift-dragging, behaves like shift-tapping an object. Also, while containers are fused, they become unbreakable units; gripping anywhere, even on members, moves the whole and not the part, thus protecting the organization.

While fused, they darken[10] and look "fuzzier", reflecting their cohesion and the indiscriminability of the space within.

Evaporating. The counterpart of fusing is *dissolving* or *evaporating*, which turns a container completely space-like at large scales. A container's role as a movable object

[9]The fusing threshold depends on the pixel-size of its members, regardless of the container's dimensions. This gives containers of all sizes the same "freezing point".

[10]Fusing increases contrast with the background, to make the container look less like space and more solid.



(a) Fused: behaves like solid

(b) Normal: hybrid behavior

(c) Evaporated: behaves like space

Figure 3.19: The three "phases" of a container

concerns its spatial relation to other objects around it. At very large scales (zoomed in), these relationships can't be seen within the view and shouldn't be manipulated blindly. So a close-up[11] container becomes completely transparent and immobile; dragging creates a selector (within that scope) as it would on the background, and shift-tapping between items toggles nothing.

Between the small-scale "solid" state and the large-scale "vapor" state, containers use the default hybrid ("liquid"?) behavior of the first-generation implementation. The inconsistencies remain, but the conditions most likely to trigger them are less frequent.

3.3 Navigation: Zooming and Panning

One of the recurring themes in information visualization is the tradeoff between *focus* and *context*: sometimes we need specific information in detail, and sometimes we need a broader but shallower overview.

One general strategy for balancing these is demonstrated by various fisheye

[11]The evaporation threshold depends on the dimensions of the container relative to the view. While containers have a common freezing-point, they may have different "boiling-points". It's even possible that a large container will freeze and evaporate at the some point, skipping the intermediate default state altogether.

views [26, 60, 47]. Generally speaking, a fisheye view is one in which certain parts of content (the focus) get a disproportionate share of the display. A universal problem is how to reconcile and "glue" together areas of different resolutions. For example, the focus might be a sharply-bounded frame or window while the context is displayed elsewhere, or the focus might be inset within the context with some strategy of "warping" to make their edges meet.

An alternative solution is a zoomable interface such as Dynapad and its Pad++ predecessors [7]. Zooming avoids the spatial gluing problem by using the same display space for both focus and context, but at different times, by *serializing* these two display roles. Zooming has instead a temporal gluing problem: how to maintain coherence over time, between views of different resolutions. Although there are many effective examples of discrete zooming, the Pad++ lineage embraces continuous zooming, where changes from one view to another are bridged by smooth animation which helps the viewer reorient.

Fisheye views have their own temporal coherence problem whenever the focus changes and the display rearranges. And it is not even exactly true that fisheye views let us see focus and context at once: we still must serialize our *attention* between them. What they support well is the *glance*: a very low-cost shift of attention from focus to context or the reverse. As we consider various implementations of zooming next, we should keep as a central goal this affordance of low-cost *glancing*.

Zooming Basics

All the implementations of zooming discussed below share some basic principles. The Dynapad window can be thought of as a rectangular viewport at some distance from the workspace surface. Zooming does not change the window size, only what it depicts. Zooming in by a factor of two doubles the apparent width and height of objects, and shows one-fourth as much area as before.

A zoom in or out is always centered around a focus point somewhere within the window (Figure 3.20). The determination of the focus point depends on the implementation, but in general it starts at the cursor. Zooming effects can be illustrated in several equivalent representations, as in Figure 3.21. From the viewer's perspective (view-centric

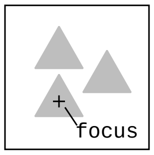


Figure 3.20: View before zooming

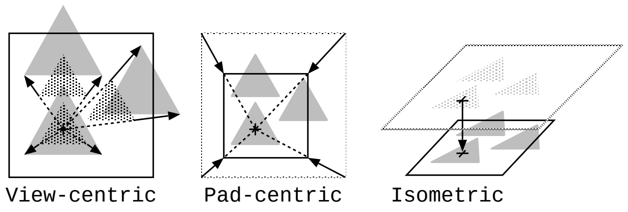


Figure 3.21: Three depictions of zooming into focus point

depiction), when zooming in, all visible points expand radially outward from the focus, and when zooming out, they contact radially inward. The farther a point is from the focus, the faster it moves. In a pad-centric interpretation, the corners of the viewing frame contract or expand relative to the pad surface. In some implementations, the focus may itself move during a zoom, essentially panning (translating) the surface at the same time. In this case, the apparent motion of a point will be the sum of its radial motion plus the translation motion. An isometric or "space-scale" ([27]) depiction makes it easier to see any translation component.

3.3.1 Implementations of Zooming

Dynapad's zooming has undergone four generations of implementation, gradually refining the cost structure of interaction. This course of this development is discussed next.

First-Generation (Classic) Zooming

In its first implementation, inherited from earlier Pad++ systems, zooming uses two separate and dedicated buttons, one for zooming in and the other for zooming out. Both directions zoom at a constant predetermined velocity while the button is held and stop when it is released. The zoom focus takes the location of the cursor when either button goes down and remains there throughout the zoom.

This design is a natural response to the poverty of input. A zoomable view into a 2-D workspace has three independent and continuous dimensions (X, Y, and Zoom). But, as discussed in 3.2.1, a mouse has only two continuous variables. Preserving the independence of zooming requires using the only remaining independent input channels: the buttons. Ideally they would emulate a continuous variable, but at minimum they must produce at least three values: zero, positive, and negative zooming. Since each button has only two values (pressed or not), three values require two buttons: one for each direction and neither for resting. Unfortunately, this mapping throws away a fourth input value: both buttons together. Since independent inputs are so scarce, that waste is not trivial. Later (in 3.3.2) we'll consider some ways to make use of that combination without interfering with normal zooming.

This implementation has a number of weaknesses. Not only does it consume (and partially waste) two valuable buttons, those buttons are arbitrary: there is nothing about these input actions that resembles or differentiates their functions. Such an arbitrary association is more difficult to learn and remember.

The need for constant velocity is problematic. If the velocity is too slow, a contextual glance becomes prohibitively expensive. If zooming is too fast, it becomes difficult both to control and to follow: users may become disoriented if they cannot visually track the sudden change in view. It might help to use "slow in, slow out" (SISO) animation, keeping visual changes slow at the start and end of a motion to help the view follow it. But in this implementation, zooming is change-oriented rather than goal-oriented: the system has no way of knowing where the user intends the zoom to end, so it cannot know when to begin slowing. Even if SISO were possible, it would disrupt the linear relationship between zoom distance and button-press time, which can be a useful heuristic for controlling zoom distance.

Finally, this style of interaction makes navigating unstable, since the objects of interest tend to drift out of view. When zooming in, for example, all points diverge away from the focus. If the user's aim is not perfect (and it never is), the intended focus will be some small distance from the actual focus, and this distance will increase, so the intended focus tends to creep out of view. Zooming *out* has a related problem: if the focus is not exactly centered in view, the viewable area tends to drift away from the objects of regard. In either direction, zooming requires continuous monitoring of progress and frequent stops to re-focus. This instability also makes it difficult to return exactly to a previous view. Quick, reversible glances between focus and context are impossible.

A partial solution: Guided zooming

Interpreted another way, this instability arises from treating the navigational space as isotropic, where any focus or direction is as good as any other. An off-center object is considered no worse than a centered object. A better system would offer some guidance or "force feedback" helping the driver orient to likely targets.

One such solution is "Leylines" [36, 38, 37]: invisible "channels" within the space which attract and direct the view toward objects. Several variants are possible, but the basic strategy has two components:

1. Automatically redirect the zooming focus to the center of the nearest object. We can think of the space being divided into "attractor basins", one per object. All clicks initiating zooming would be treated as if they were at the focus of that basin, the center of its object.

2. Add a panning component during zooming to move the focus toward the center of the view.

The overall effect is that zooming in anywhere draws the nearest object toward center of the view, giving the impression that the view is somehow "on a track" rather than being a free "vehicle" in isotropic space. Zooming out can be made to reverse this trajectory and head back toward the previous view.

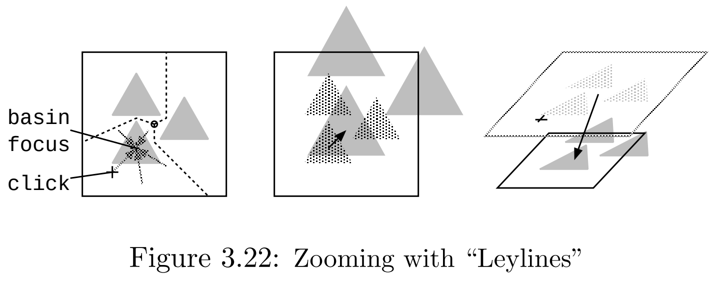


Figure 3.22: Zooming with "Leylines"

Such guidance has considerable cognitive benefit. Aiming can be relatively sloppy and still get to the target. Additionally, the zooming task becomes nearly *ballistic*

[42]: once the initial targeting decision is made, the action guides itself toward the intended state. No course-correction is required, and the monitoring of progress is much simpler, since the only remaining decision is when to stop.

Second Generation: Interpolated Zooming

Dynapad's second implementation of zooming uses a similar technique. Zooming is (usually) directed toward an object, which simplifies the interaction. But Dynapad's implementation also addresses the other problems of two-button zooming mentioned above.

The original justification for two-button zooming was to preserve the independence of three dimensions of input. But this turns out to be unnecessarily restrictive. Inputs need to be independent only when used concurrently. It might be possible in principle, for example, to move an item and zoom at the same time. But in practice, such multi-tasking is infeasible since either activity alone requires considerable attention. The zoom focus uses the mouse position, but only up until zooming begins, since the focus remains fixed during zooming. Therefore the cursor position can be overloaded for zooming, offering a third continuous variable and eliminating the need for two buttons.

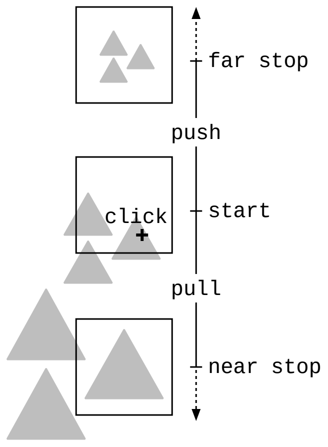


Figure 3.23: Interpolated zooming

With *interpolated zooming*, the user holds down a single (the middle) button, and the mouse's vertical axis is reassigned to control depth: pushing away (up) zooms out and pulling (down) zooms in. The view is interpolated between three points:

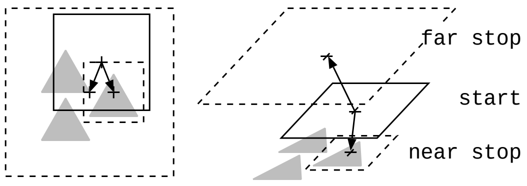


Figure 3.24: Trajectory of view center

- The starting mouse position corresponds to the starting view;
- The bottommost position ("near stop") gives a close-up[12] on the targeted object;
- The topmost position ("far stop") always gives a view of the entire workspace, no matter what is targeted.

The two stop positions in the mouse's motion are chosen arbitrarily and determine the sensitivity of zooming in each direction. Zooming ends at the stops even if the mouse is moved beyond them and reverses when the mouse crosses them in the other direction, eventually passing through the start view again toward the opposite stop.

The Zoom Target. The target object is determined by the initial click, following the same rules used for dragging (i.e. target the topmost object under the click, which may be a container). As with Leylines, the zoom focus becomes the center of that target, so the target is centered in view at the near stop, at a scale where its maximum dimension fills a fixed percentage (the "close-up ratio", e.g. 80%) of the view.

If the target object is part of a selection, the entire selected set becomes the target, with a focal point at the center of the selection's bounding box. This function lets selection specify an arbitrary scope for zooming, just as with other operations (see 3.2.2). Viewing becomes a postfix operation (*view[x]*) like any other.

If there is no object under the initial click, the focus takes that point on the workspace surface and the zoom depth stops at a predetermined multiple of the start depth. Such "free" zooming usually proceeds in several pulls (or pushes), as the user decides gradually where to view.

[12]Hitting the near stop also triggers a high-resolution view of the target (if an image); see 3.1.1.

It is possible to override the stops; currently this is done by holding the Shift key (overloading its *invert selection* meaning with the other button). Bypassing the stops allow very close-up or distant views.

Advantages of Interpolation. One advantage of interpolated zooming is that the user can use fine motor control of position to control velocity. This lets them emulate manually a "slow-in slow-out" motion whenever needed for coherence but also move as quickly as desired in mid-zoom. Additionally, because there is a predetermined stopping point (the targeted object), SISO or any other non-linear trajectory can be included automatically in the interpolation, so that zooming slows down at the start and end even if the mouse moves at a constant speed.

Interpolated zooming also offers a simple and universal recovery from getting lost: no matter what the current view is, even if no objects are visible, simply pushing until the automatic far stop is reached will return to a familiar "home" view.

Furthermore, because the near stop is also automatic, zooming can be *completely* ballistic: once a target is chosen, no cognitive effort it required to reach it. There is no need to monitor progress and decide when to stop, so the motion can be much faster than attention can follow. All that is needed is a sufficiently long push or pull, a gesture which is very easy to master. This makes is very easy to "glance" at focus or context.

A glance in either direction is also reversible: as long as the button is held, returning the mouse to the starting position recovers the starting view. One flaw, however, is that there is no stop at the start position. Therefore some cognitive effort is needed to monitor the return and stop at the correct position.

Bent Trajectories. An additional problem is that zoom trajectories may be *bent*, as Figure 3.24 demonstrates. If the center of the starting view is not collinear with the center of the near and far views, passing through the starting position suddenly changes direction and can be disorienting.

An even worse cases arises if the depth of the starting view is beyond one of the stops instead of between them. This can happen if the override (*Shift*) is used to zoom very far out, as in Figure 3.25, or very close in. In such cases, passing through the

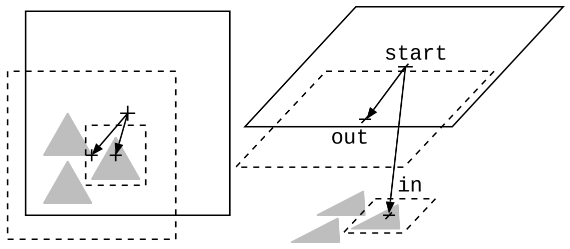


Figure 3.25: Strongly "bent" zooming trajectory

starting view will not only bend the trajectory but actually reverse direction: pushing will first zoom closer, then farther away. Although the motor actions needed to reach the near and far stops are the same, the visual feedback is inconsistent and disorienting.

Third Generation: Gated Interpolation

The third implementation of zooming addresses both the problems of bent trajectories and the stop-less starting view. Both result from the same design choice: allowing zooming to pass the starting view in either direction. In fact, there is no need to allow this: a single glance is only meant to go in one direction, toward context or detail, but not both at once.

Gated interpolation just makes a small modification. It allows zooming in either direction initially, but once zooming has gone a certain distance (through a "decision point" or "gate"), it blocks the opposite direction and turns the starting position into a *middle* stop. If zooming begins outward, the view stays between the start and far stop. If zooming begins inward, it stays between the start and near stop. A single zooming action can be in or out, but not both.

This prevents any confusing bends in the zooming path. It also provides a stop at the starting view, so that recovering from a glance is ballistic. If there is some need to zoom both in and out from the starting view, it only requires bumping up against the start, releasing the button, pressing it again to start a new glance, and zooming in the other direction.

Losing the Focus. One flaw in the current implementation is that the cursor always follows the mouse motion – which means it often diverges from the prior focus. And

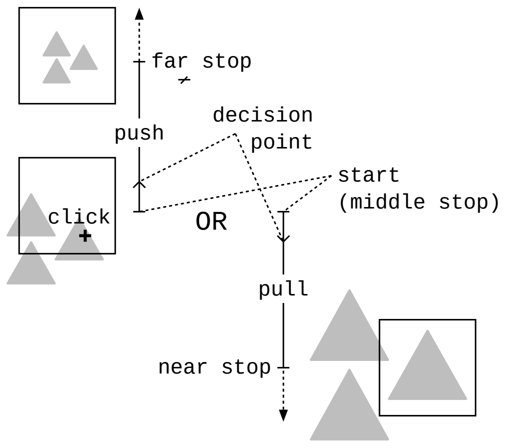


Figure 3.26: Gated interpolation zooms in or out in separate actions.

once the button is released, the old target is forgotten. The cursor must be manually returned to the target for re-zooming.

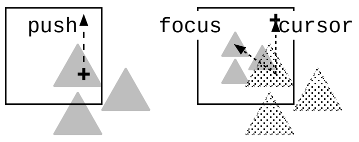


Figure 3.27: The cursor may diverge from the focus during zooming.

One of the situations where this presents a problem we might call *post-ascent disorientation*. Often users will look closely at an object, then zoom out to move it to a new location. But if the zoom-out is done too quickly (and it can be done instantly with a quick push), it's easy to lose track of the object that was just viewed. Keeping the cursor locked to the object during zooming would reduce such disorientation.

Fourth Generation: Multi-stage Zooming with Implicit Cohesion

So far, all the variants of guided zooming have considered only individual objects as targets. But this ignores the importance of more complex, multi-scale structures, such as containers (3.2.4).

For example, consider the container in Figure 3.28. Sometimes we want focus on the container as a whole, sometimes on a particular item inside it. Where should the zoom direct us? This is similar to the multi-scale ambiguity when moving the container or contents, and it's resolved the same way: the scale used depends on where the cursor is when the drag begins. Pulling on the container body, or any part when it's fused, zooms to the container itself. A second pull can then zoom to a particular object. This is a simple example of *multi-stage* zooming, one which emerges naturally from the physics already in place.

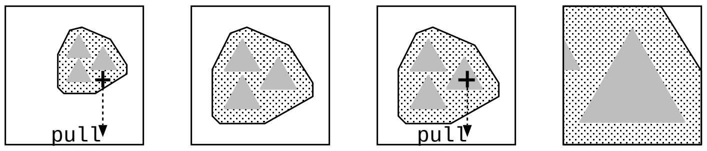


Figure 3.28: Zooming in two stages: first to a container, then to its member.

Consider a different type of complex structure, an implicit grouping of nearby items. There is no explicit container to grip and zoom into directly, but if the group is first selected, then pulling on one of the members will zoom to the entire selection. However, in third-generation zooming, there is no second stage: pulling on a member will continue to target the entire group. Furthermore, zooming into the selected set only works if one of the items is targeted. Gripping the space between items is considered untargeted "free" zooming, which moves closer but doesn't center or stop at the selection.

But implicit groups often have a *proximity gestalt*, further accentuated by selecting them, which suggests a coherent "chunk" like an explicit container. This creates an expectation that they behave in the same way during zooming. To maintain this parallel, Dynapad's fourth-generation zooming introduces *implicit cohesion*: gripping and zooming anywhere within a selection treats it as a coherent unit, as if it were bound by an explicit container.

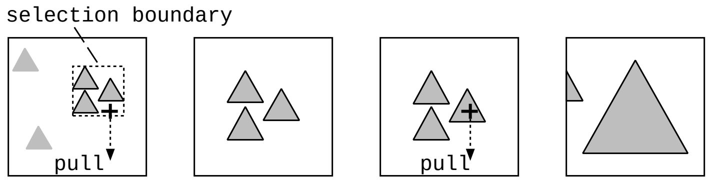


Figure 3.29: Multi-stage zooming into selected implicit group

The primary difference in this implementation is how it deals with selections of multiple objects, such as an implicit group. Targeted selections are treated differently in two ways:

1. If the current view is not equal to the "near stop" view (centered and close-up) of the selected set, it zooms there, as if the set were a normal object. If the view is already at the near stop, pulling on any member of the set re-targets and zooms to that member.

2. Gripping anywhere within the bounding box of a selected set is equivalent to gripping one of its members when first pulling the set into view. That is, pulling amidst an implicit group is like pulling on the body of an explicit container (see Figure 3.29).

These two changes give implicit groups almost the same behavior as explicit containers.

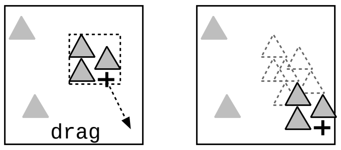
Implicit Cohesion in Zooming vs. Dragging. The problem with implicit cohesion for zooming is that it creates an expectation of a similar cohesion for dragging. That is, dragging amidst a selected set could be expected to move the implicit group, just like dragging between members of an explicit group (on the body of a container) moves it (when it is solid-like, at least). As Table 3.1 shows, three of the four combinations have cohesion, making the fourth seem anomalous. Indeed, users have made errors from and commented on the inconsistency.

Table 3.1: Implicit cohesion is inconsistent for dragging.

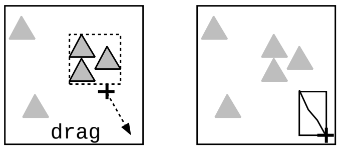
Cohesion when...	Zooming	Dragging
Explicit (container)	X	X
Implicit (selection)	X	-

However, completing the pattern by enabling implicit cohesion for dragging would introduce an even more troublesome inconsistency: empty space itself would react differently depending on whether a drag is inside or outside a selection's bounding box. Dragging inside would move the group, but dragging outside would deselect it and start another selection. The border between these behaviors would be abrupt and invisible, as Figure 3.30 illustrates.

Implicit cohesion in zooming has a similar discontinuity at the boundary of selections, but the effect is much milder: zooming outside the boundary will still zoom



(a) Dragging just inside boundary would move set



(b) Dragging just outside boundary would re-select

Figure 3.30: Implicit cohesion for dragging would have sharply discontinuous behavior.

in but on a somewhat divergent path, which can be reversed and corrected with little effort.

Direct (Double-Click) Zooming

One final technique for zooming has been available since early in development, but can better assessed now that we've considered the cognitive implications of interpolated zooming. Instead of dragging with the zoom button down to approach the target, the user can double-click with the primary (move/select) button to zoom directly to the target. If the target is an image, this also triggers its high-resolution version (see 3.1.1).

3.3.2 Panning

Zooming's counterpart is *panning*, translating the view around the workspace without changing scale.

Panning vs. Scrolling. Panning has a ubiquitous analogue in other applications: *scrolling* a normal document in a window. However, in Dynapad the "document" (the workspace) extends infinitely in all directions. It would be reasonable to treat Dynapad's *working area*, the bounding box of all objects, as a gradually expanding domain, and display scroll bars at the window's edges to situate the current view within that domain. However, it's not clear that this affords any utility that Dynapad doesn't subsume in other ways.

The utility of scroll bars is twofold:

Manipulation: One can slide the bars (or press "arrow widgets") to pan one axis at a time, but this requires moving the cursor to the margin, away from the objects of interest. It's far less work to have an effective *local* control of both axes at once, which panning offers.

Feedback: Even without manipulation, scroll bars show one's position in the world, to prevent becoming lost. But Dynapad already offers an even better alternative: a quick glance outward to *see* the whole world and an easy return to the "local" view.

Velocity- vs. Position-coupling. The obvious solution for local control is to harness the two continuous input variables of the mouse (or table) device: to pan by dragging with some (to be decided) button pressed. An important question is then whether to couple that drag vector to the *velocity* or *position* of the view center. Many applications use velocity, which has the advantage of potentially fast travel over distances much greater than the view's width. But as with zooming, velocity-controlled panning risks being either too slow to be effective or too fast to control, and it requires continual monitoring while in transit. Position-coupled panning, however, has the advantages of interpolated zooming: fine motor control with instant arrival when desired.

View-dragging vs. Ground-dragging. The next design question is whether a panning-drag should move the *view* or the *ground*, whose relative motions are in opposite directions.

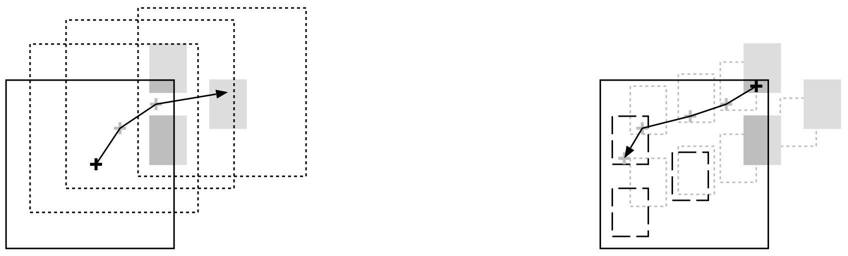

(a) Dragging the view (b) Dragging the ground (Dynapad)

Figure 3.31: Variants of position-coupled panning

Each choice has advantages:

View-dragging: The cursor remains at the same position in the view but moves relative to the ground. Because the cursor is always in view, a single pan action can travel any distance. However, there is nothing to stop the view in a particular location, so the user must monitor progress.

Ground-dragging: The cursor moves across the view, but stays anchored to the ground. Panning stops when the cursor hits the view's edge. Therefore a single pan cannot travel farther than the width of the view; longer travel requires repeated dragging ("pawing"). However, the ground-anchor can be used to target an item or location, and the pan will always stop with that target in view. Therefore the travel can be ballistic, requiring no monitoring, and can be done very quickly.[13]

In general, view-dragging is more efficient for long distances, but ground-dragging is better for short distances. But panning is intended primarily for short distances, since long distances can be traveled easily and reliably by zooming out and back in. Therefore Dynapad uses ground-dragging to pan.

A consistent metaphor. This choice has another advantage: the ground-dragging metaphor is consistent with zooming. When zooming, pushing zooms out, moving the ground away, and pulling brings the ground closer. So with both zooming and panning, holding the *view* button is like *gripping and dragging the workspace surface*, just like holding the *manipulate* button grips and moves objects.

This metaphor has another convenient facet: panning's "up" axis becomes zooming's "up" axis if it pitches ninety degrees backward (away from the viewer). This would be done physically by pushing on the top of the mouse — where the button is. So the physical action of pushing the button for zooming is similar that of tilting a 3-D controller emulating the three dimensions of the navigational space. This coherent spatial metaphor is less arbitrary and presumably easier to learn than two buttons or even a consistent view-dragging metaphor.

[13]Of course, *centering* the target still requires monitoring. A panning-drag could be made to stop with the target centered, like zooming, but this would cut in half (from view diameter to radius) the range of a single stroke.

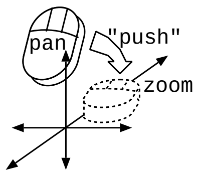


Figure 3.32: “Tilt” metaphor for zoom/pan overloading

Integrated Zooming and Panning

Because zooming and panning are so closely related, it is fortunate and appropriate that they can be overloaded to use the same (middle) mouse button. This button then has the consistent meaning of controlling the *view* (or of *gripping the world*, as discussed above).

It is easy to see that panning horizontally will not interfere with zooming, which uses only the vertical axis. But vertical panning can also be accomplished with a “gearshift” gesture: to use the vertical axis for panning, first “jog” the cursor sideways.

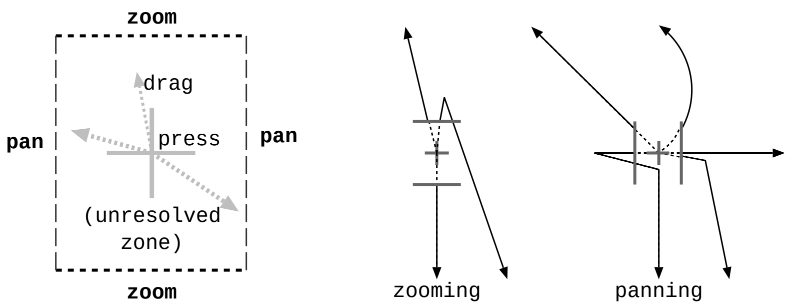


Figure 3.33: Integrated zooming and “gearshift” panning

When the *view* button goes down, a small imaginary box[14] is constructed around that point. Until the cursor leaves the box, the action is ambiguous, and neither zooming nor panning begins. But a decision is made when the cursor first crosses an edge. If it crosses the top or bottom edge (i.e. its motion is initially vertical), Dynapad enters *zooming* mode: any further horizontal motion is ignored, and the view begins gated interpolation as described above (3.3.1). But if the cursor crosses a side edge (i.e. its motion is initially horizontal), Dynapad enters panning mode: any further motion on

[14]The box is taller than square so that a diagonal reliably pans instead of crossing the decision point.

either axis drags the ground. Zooming and panning cannot be done in the same action, but they can be alternated by releasing the button after each.

Naturally, there is some potential for error in confusing vertical panning and zooming; we have observed this error frequently with many participants. Fortunately, the cost of the error is low; unwanted zooming can be easily undone by returning the drag to the middle stop.[15] In our experience, most operators quickly learn reliable techniques for panning vertically without zooming. Effective actions include "arcs" (Figure 3.34(a)) and "shaking loose": rubbing horizontally to prevent zooming, then panning straight vertically (Figure 3.34(b)).

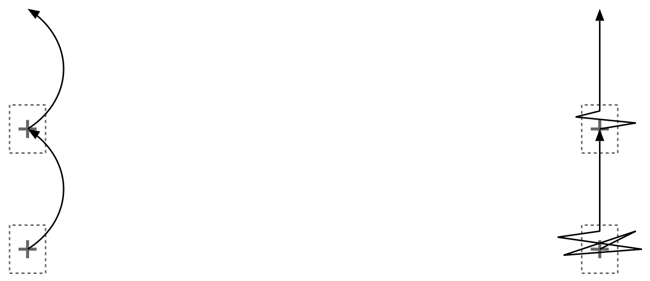

(a) Successive arcs (b) "Shaking-loose" from zooming

Figure 3.34: Some vertical panning techniques

Synthetic Discrete Panning

The physics of zooming and panning described here allow a convenient but unplanned trick: long-distance, guided panning can be synthesized by using zooming to follow "trails" of adjacent items. Remember that targeted zooming includes a translation component, re-centering the target. If the initial zoom scale already equals the final scale, only the translation component remains, and "zooming in" simply pans to center the new target. If there is a line of adjacent targets, repeated pulling will "ratchet" the view along that line. The process is also fully reversible, reducing any concern of getting lost while "traveling" at a close-up scale.

[15]Unwanted *panning*, although less frequent, is less easy to revert. This could be improved with "gated" panning.

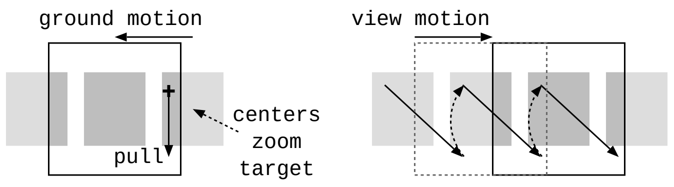


Figure 3.35: "Ratchet" panning by iteratively zooming to adjacent items

This technique requires very particular conditions. First, nearby objects must have the same maximum dimension so that they have the same stopping scale. Second, the neighbors must be close enough together at the stopping scale that when one is centered the next can be seen and targeted. This requires a coordination of the objects' separation and aspect ratios, the viewing frame's aspect ratio, and Dynapad's close-up ratio.

As it happens, these conditions are satisfied by the initial layout grid (see 3.4.1) of size-normalized portraits in a full-screen view. One of our users discovered this and began using it to traverse rows and columns of the collection at a close-up scale. Manual horizontal panning would have worked as well, but apparently this participant liked successive items to "lock-in" automatically.

This behavior illustrates not only the utility of *serialized* arrangements (see 2.2.2) but the need for a low-cost procedure for iterating through them when visual bandwidth is limited. This feature emerged in Dynapad by accident; future designs should include deliberately a more robust equivalent.

Carrying while Navigating

Selecting and moving objects and controlling the view constitute the "whitespace" of collection management. Our goal was to make them as easy as possible, ideally requiring only one hand. Dynapad's latest design manages to overload all these functions[16] into only two buttons: one for *manipulation* and one for *navigation*.

This arrangement offers an additional important convenience: because these two buttons are independent, navigation can be done while carrying objects. Once an object is gripped, it follows the cursor as long as the *manipulation* button is held.

[16]Incremental selection, of course, requires an additional input.

Meanwhile, the *navigation* button can be pressed and released to zoom and pan while still holding the object. This makes it much easier to transport objects over large distances. Figure 3.36 illustrates an example with panning.

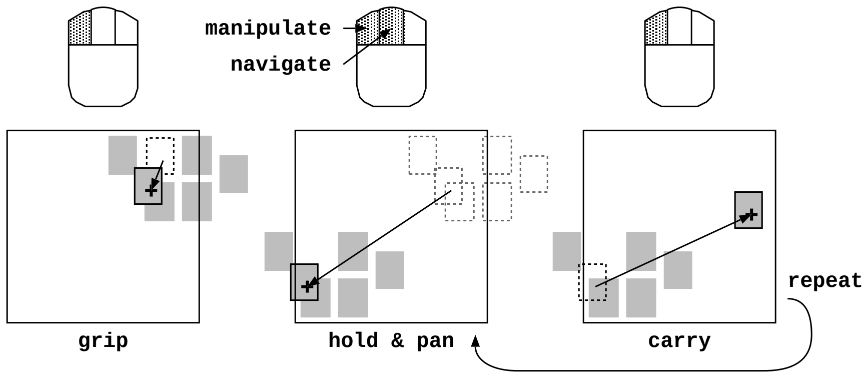


Figure 3.36: Panning the view while carrying an object uses both buttons at once.

Re-orienting by grip. This ability to grip objects while navigating affords another unplanned but valuable strategy: reducing post-ascent disorientation by "keeping a finger" on an object. After viewing the target close-up, first grip it for moving, and only then zoom out. Even if lost during the zoom, it's still in hand, ready to be moved, and its motion re-orients the user's attention to the prior context.

3.3.3 Additional Operations

All additional operations of Dynapad can be called from a hierarchy of pop-up menus, which are invoked by the third mouse button. However, since we hope to make Dynapad require only two buttons, we are exploring alternative actions by which the user can pop up these menus.

3.3.4 The DiamondTouch Table Interface

In addition to the Dynapad's normal mouse-driven interface, we have begun exploring the use of alternative input devices. Foremost among these is the DiamondTouch (DT) table from Mitsubishi Electric Research Labs [17]. The DT table features a grid of short-range antennae which detect the points at which a user touches the table surface. When Dynapad's display is projected onto that input surface, it gives a user an engaging feeling of directly manipulating the projected virtual objects.

However, we needed to modify Dynapad's controls to accommodate the table's more restrictive input space. Instead of three buttons, the DT table effectively has *less than one.* If we think of a single finger on the table as a one-button "mouse", then it generates a position only when the "button" is pressed. That is, while a real one-button mouse has $x \times y \times 2$ states, a single DT touch has only $x \times y + 1$ states.

Of course, a DT user can use multiple fingers at once, which is potentially more expressive. However, although separate DT users generate independent signals, multiple touches by the same user create ambiguity, as Figure 3.37 illustrates.

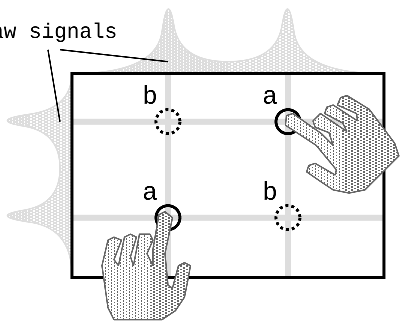


Figure 3.37: Multiple touches on the DiamondTouch table are ambiguous: pairs **a** and **b** both produce the same signal.

Our solution is to eliminate the ambiguity of a second touch by restricting its freedom of position. We dictate its location (x_0,y_0), then attribute any signal with either of those components to that "auxiliary" touch and not to the "main" touch, which we can then infer to be at a different coordinate (x,y). The best location for (x_0,y_0) is in one corner, which leaves the main work area slightly smaller but uninterrupted. In short, we emulate a second button by stealing one edge from each of the continuous input variables (x and y), as Figure 3.38 demonstrates.

This second button (B2) serves as Dynapad's navigation button, replacing mouse button-2 in all the panning and zooming actions described in this section. It requires the user's non-dominant[17] hand, since the dominant hand must simultaneously control the navigation point (B1).

[17]Naturally, for left-handed users the button should be moved to the right side.

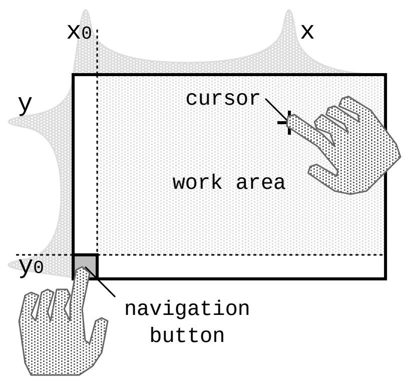


Figure 3.38: Emulating an unambiguous second button by stealing a corner of the input space.

Menu Items. Even with a second button, we still need a replacement for the menu items triggered by a third button, B3. It would be easy enough to put B3 in line with B2, along the bottom or left edge. However, using Dynapad's normal pop-up menu system with B3 would cause interactions between B3 and B1, the cursor position.

As a solution, we bypass the menus altogether and instead assign a separate button directly to each of the most-used menu items, which do not need a concurrent touch at B1. We put nine such buttons along the bottom edge: one to create each of four region-tool types (clump, tray, lens, stamp), copy, unlock/lock portrait, launch PDF viewer, undo, and redo.

Any other needed operations could be done with the mouse and keyboard adjacent to the table — inconvenient, but better than nothing. During our study, the interviewer used these whenever the subject directed, considerably reducing their usage cost.

Note also that, unlike pop-up menus, these continually visible buttons serve an additional function of *reminding* an inexperienced user what actions are available.

Clearly this prototype DT interface is not optimal, but it satisfied our immediate research goals. However, it also introduced some subtle changes to Dynapad's cost structure, which manifested themselves in the subject's activity, as we'll see in Chapter 4.

3.4 The Cue Structure of Collections

In paper collections and in Dynapad, spatial arrangement is an important source of expressiveness. Expressive arrangements improve organizational clarity, making more visible both the overall structure of a collection and specific relations between items.

But Dynapad is not limited to the affordances of paper; it can express these relationship in ways besides spatial positioning. This section examines various ways by which Dynapad tries to improve collections' organizational clarity.

3.4.1 Importing a Collection

So far we've considered scenarios in which a user already has an collection in Dynapad. But how does a collection get started? Where do the documents come from?

Currently, all documents enter Dynapad by being *imported* in batches from directories in the filesystem. With each batch, the user can choose to import a single directory or an entire tree (i.e. descend recursively into all subdirectories). We can regard the single-level import to be a trivial special case, and in the discussion ahead assume a more complex hierarchy.

Importing a batch has three steps. First Dynapad filters the selected directories to accept only PDF documents (or various image file formats, if the user chooses). It then creates a default portrait (see 3.1) for each PDF or image file. Finally, it offers the user a rectangular "footprint" of the incoming batch to place somewhere in the workspace, and then arranges the new portraits in the chosen location.

Pandora's Shoebox. Various studies of people's collection-management habits, particularly with photographs [57, 31, 30, 72], suggest that people invest relatively little effort up front in organizing their collection. Nevertheless, they may *feel* as if they are more organized, particularly with digital collections, when their entire collection is reliably available in one place [57, p.411]. They are content to cache their collections into essentially a "shoebox", a container which secures the set but does little to promote or preserve any internal organization. Specifically, a literal shoebox affords poor visibility and stability, and therefore has a potentially acute instability-unclarity cycle of disorder (Figure 2.10).

The same cost structure exists more generally for digital documents on a computer file system: many people settle into a habit of dumping all personal documents into a "shoebox" directory[18] — and quickly slamming the lid shut, increasingly unwilling to confront the disorder which accumulates there. A directory does provide some automatic structure: sorting by name or date for example. But naming items effectively requires effort, and so increases the pressure to defer categorization and enter the cycle of overload depicted in Figure 2.12 — without the dampening influence of visibility. For many of us, a filing system turns quickly to a fearsome shoebox. It has structure, but is not necessarily to be trusted.

The primary challenge of importing is to satisfy a dilemma: we want to preserve and accentuate meaningful structure, but suppress meaningless structure which has emerged only from disruption and overload.

Initial Layout

The initial layout of a batch is potentially very important, especially if a collection is imported in a single batch. The first imported batch dictates the state of the user's artifact (the workspace) upon which they begin the cycle of reflective interaction.

Remembering the cognitive benefits of stacks, we'll assume the initial layout should be *visible* (unoccluded), *serialized*, and *contiguous*. A grid has these properties. We also assume its maximum dimension (width or height) should be minimal so that the resolution is greatest when the full layout is visible. Therefore, by default we arrange the batch into the smallest square matrix that can fit all the items.

However, the question remains how to represent the incoming *directory structure*, if at all. Dynapad has evolved through three solutions.

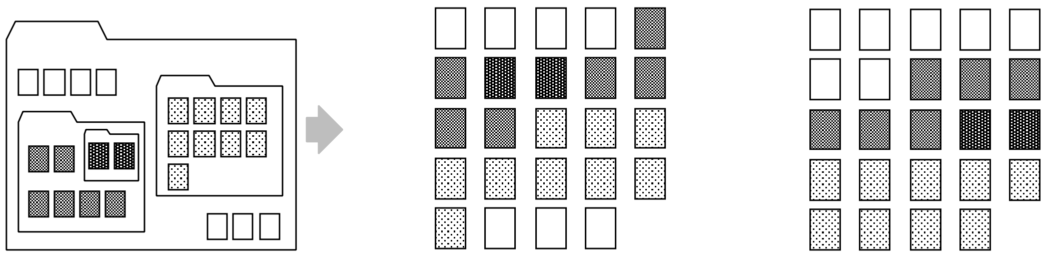
Single block, flattened structure. The first and easiest variant simply ignored the directory structure, "flattening" the entire incoming batch into a single square block, as in Figure 3.39. Items within a directory should presumably be sorted by filename (or perhaps date), but how should different levels be integrated? Possibilities include:

[18] Indeed, some systems even encourage this with a default "My Documents" folder.

Mixed: Items from all levels are mixed and sorted, and directory names are ignored. Items are strictly ordered, but their grouping by directory is completely obscured, and initial neighbors may be widely separated. This may be appropriate if filenames are systematic and memorable or if directories are meaningless.

Depth-inserted: Subdirectories are inserted by name. Filenames may be out of order between levels, but directories remain semi-coherent (branches will not be mixed).

Depth-last: Within each level, all files precede subdirectories. Order is the least consistent, but all directories remain contiguous.[19] This variant may be best when names are less important than structure.



(a) Directory structure

(b) Flat grid, depth-inserted

(c) Flat grid, depth-last

Figure 3.39: Importing and flattening a collection

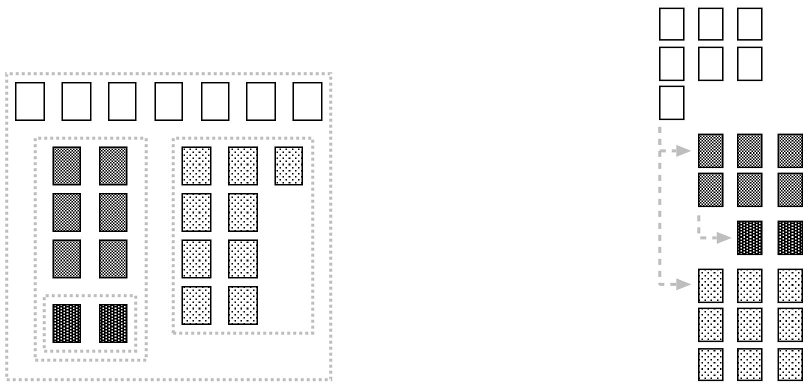
Which solution is best depends on the user's investment in and understanding of the file collection's organization and namespace, which we cannot guess.

Directory blocks

Another class of solutions is to structure the layout hierarchically. Instead of a single block, each directory can be blocked separately and the blocks arranged to mirror the directory structure.

There are at least two variants of this approach:

[19]In all variants, of course, spatial contiguity is interrupted across a line wrap. This is especially notable when the intended group is shorter than one row and its two parts are spatially isolated.



(a) Nested: layout is difficult and inefficient for complex hierarchies.

(b) Outline: layout is consistent for all hierarchies.

Figure 3.40: Importing and preserving directory structures

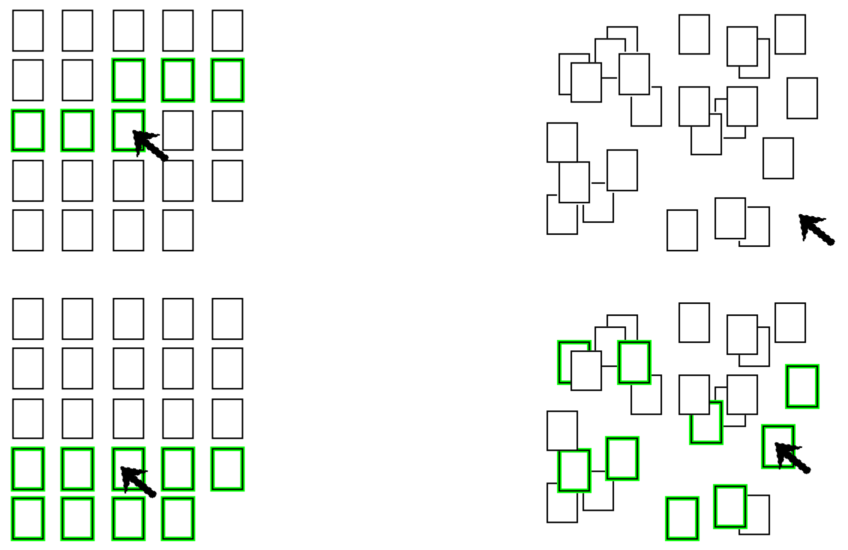
Nested: Each directory is assigned an area big enough to contain both a block of its files and the areas of its subdirectories (Figure 3.40(a)). This has the potential advantage of keeping groups together while intuitively depicting containment. However, it is a computationally difficult problem and cannot guarantee an effective and efficient result for all hierarchies. The Quantum Treemap algorithm [8] is one possible solution.

Outline: A much simpler and universal solution is to abut each directory block in a linear outline form, with indentation to indicate containment (Figure 3.40(b)). If the resulting aspect ratio grows too long, it can be wrapped into multiple columns. Dynapad briefly used a (non-wrapping) version of this.

Both of these solutions, however, create the risk of overemphasizing structure which is merely incidental.

Single-block with "memory"

Dynapad's third implementation attempts a compromise, making the structure available but not dominant. Items are arranged in a single, unstructured block, but they *remember* their original directory. The implicit groups of items from the same directory be made visible by *linked brushing*. That is, whenever the cursor passes over ("brushes") an item (the brushing *source*), all the items from the same directory are highlighted with a colored outline (Figure 3.41(a)).



(a) Items from a common directory highlight when brushed.

(b) Directory is remembered even after items are moved.

Figure 3.41: Brushing linked items

This effect continues through the lifetime of those items (unless switched off), even after they are moved from the initial grid and dispersed around the workspace in other groupings (Figure 3.41(b)). This lets the items essentially participate in two independent groups at once: one by spatial proximity, negotiated over time, and another by synchronized highlighting, fixed initially and permanently available on demand. Directory information is present but does not bias the layout, potentially making it easier to re-envision the collection's organization as needed.

In terms of the influence graph of Figure 2.26, the organization improves flexibility and expressiveness without losing stability.

3.4.2 Generalizing Linked Brushing

With this example in mind, let's explore linked brushing as a feature that can be generalized for other uses.

The term “brushing” was first adopted by Becker & Cleveland [6] to describe a similar technique in data visualization. A coherent set of data points could be temporarily “painted” with a color in one representation in order to be tracked more easily when they were dispersed in an alternative representation. This helped the data analyst spot patterns between incompatible views. The same purpose applies here: linked brushing (by directory, for example) lets us track potentially meaningful groups through multiple spatial arrangements.

Highlighting as a Signal

There are essentially two design facets of linked brushing: *what* is brushed (the information in the links) and *how* (the visual signal which displays it). For the moment, let’s consider just the latter, the design space of highlighting. Currently Dynapad uses a colored border (Figure 3.41), but this is only one of many possibilities.

Synthetic Contiguity. Highlighting manufactures visual contiguity without spatial proximity. With a sufficiently distinctive visual marker a diffuse set of items can “pop out” as a virtual group, and the viewer’s eye can easily travel between members even with distractors sharing the space.

Retinal Properties. What visual markings will be effective for this purpose? In Bertin’s terms [10], they must be *selective*: able to be isolated by the eye to differentiate items. Color, size, and brightness all have this property. So we can imagine “highlighting” by enlarging brushed items, for example.

Ideally, highlighting should also be *associative* [10]: able to be ignored by the eye to suppress differences when the virtual grouping is disregarded. This requirement is softened when the brushing can be switched off. Neither size not brightness are associative; large or bright items are hard to ignore. Only color is selective and associative. Dynapad’s colored borders do in fact increase the overall size and luminance of the highlighted portraits, so they are not strictly associative.

One problem with color highlighting is that the portraits themselves may include the same visually salient colors (indeed, that contributes to their expressiveness and visibility). Also, closely-packed colors have additive effects: for example, adjacent red and green may appear yellow. This reduces a color signal's reliability and limits its variability to only a few hues (ideally, only the three additive primaries, red, green, and blue).

Multiple Highlights. Ideally, a portrait could have multiple highlights at once, perhaps representing second-order directory groupings or other information. Also, *selected* items are highlighted by the same mechanism (with a red border).

It would be easy simply to wrap additional borders around portraits. But this has complications: it worsens additive color-bleed, and gives outer highlights greater salience as the perimeter increases.

Currently Dynapad overlaps highlights on a single shared border. Only the most recent highlight on top is visible. This means, for example, that selecting an object (red border) temporarily hides its membership in a directory (green border) or other brushed group.

"Rubbing" for Change. Even if color and other visual parameters have limited selectiveness when *static*, they are highly selective when they *change*. Because brushing activates when the mouse cursor passes in front of objects, it is very easy to make highlighting *flicker* by "rubbing" the cursor across a portrait's edge. Such flickering is highly visible in the otherwise static view, and significantly improves brushed groups' visual coherence.

Visual Weighting by Zooming. Dynapad's colored-border highlights have a fixed width (currently one pixel) regardless of the view's zoom level. So number of pixels used by a highlight is proportional to an object's perimeter, which varies linearly with zooming. But the pixels used for a portrait itself vary by area, the *square* of zooming. This means that when the view is wide (zoomed out), and the portraits are very small, highlighting consumes a greater portion of the display — that is, the *signal strength* of the highlighting is increased, relative to the visual "noise" in the variability in the portraits themselves. So in addition to its "normal" or "pragmatic" purpose, zooming has an emergent affordance of regulating the dominance of highlighting. We might consider this an example of "epistemic" [44] zooming, and we'll see additional examples.

By design, brushing has the strongest signal when views are wide, when assessing one's collection broadly rather than focusing narrowly on one part. Its primary intent is to make visible the collection's large-scale structure to help the clarity of organization.

Brushing Other Information

Next let's think about *what* gets brushed. We've seen the example of shared import-directory. And selections, although not brushed in the same way, use highlighting to bind them together visually.

Brushing Duplicates. Dynapad also uses brushing to tie together all instances of a document. Any item that has been copied brushes (in yellow[20]) itself and its duplicates. This enables *cross-referencing* in support of *multiple classification*: an item can be put in multiple places, and the user is alerted (by the highlighting) to the others when any one is found. Brushing also links copies made automatically, such as the "phantom" duplicates in a lens (see 3.6.2).

Currently duplicates are detected and linked only when items are explicitly copied within Dynapad. Items may also have redundant but unlinked duplicates if they are imported more than once or have duplicates in the file system. Both of these types of redundancy should ideally be made visible to the user, but doing so may force a more flexible definition of "sameness" and additional variations in highlighting.

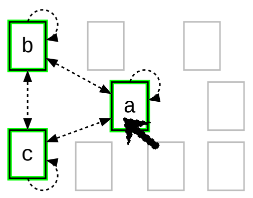
Asymmetric Brushing Relations. All the examples of brushing so far have tied together the members of *equivalence classes* (e.g. copies of the same item, or from the same directory). But an equivalence class is actually a special case of a *binary relation*, which can be represented as a directed graph with arrows between items in the relation.

In an equivalence relation, the arrows are exhaustive,[21] connecting all members of the set to all others. Figure 3.42(a) shows an example. Items a,b, and c are connected in an equivalence relation, and brushing one member (the source, a) means highlighting all the items connected to it (including a itself).

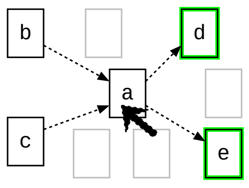
[20]In retrospect, yellow is a poor choice, since it is the product of the color bleed of the existing red and green highlights. The third primary, blue, would be better.

[21]In graph-theory terminology, an equivalence relation is symmetric, reflexive, and transitive.

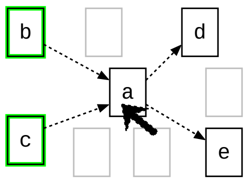
But there are many other types of information within a collection which can be expressed in less exhaustive relations. For example, documents may *cite* one another, a relation which is (usually) asymmetric. In this case the arrows flow only one way. Brushing a document's citation *outflow* would highlight others which it cites, as in Figure 3.42(b). Similarly, brushing its *inflow* would highlight documents which cite it (Figure 3.42(c)). Inflow can also be treated as the outflow of an "opposite" relation.



(a) Brushing an equivalence class



(b) Brushing citation outflow (**a** cites **d**,**e**)



(c) Brushing citation inflow (**b**, **c** cite **a**).

Figure 3.42: Examples of brushing relations

Simultaneous brushing. We can then imagine *multiple* arbitrary relations being brushed at the same time. One problem is that of coordinating multiple highlights such that each remains retinally selective. With two or more relations brushed at once, it becomes increasingly difficult to isolate individual dimensions. In particular, multiple-brushing likely negates the use of "rubbing" to flicker highlights. Flickering is not associative; it cannot be easily ignored to consider only color. Therefore it is likely to make the *union* of a source's outflow a more salient grouping than any of its individual relations. Nevertheless, future versions of Dynapad may find a way to let users coordinate multiple brushed relations effectively.

As a technical matter, Dynapad already includes a *brushed-relation* object class which automatically highlights a brushed source's outflow for any relation created by another component of the software. For example, the importing module creates a "same-directory" relation. Currently no modules create relations for other aspects of a collection (like citations), but they can be added relatively easily.

Selecting and Gathering Brushed Relations

We considered earlier that selection allows non-contiguous arrangements to be cohesive, so that they can be moved without disrupting their relative structure. Such cohesion is transitory: a diffuse set must be re-identified and re-selected each time it's moved.

Brushing helps by remembering and reminding of certain discontinuous sets, and improving their visual continuity even when they are spatially dispersed. But brushing alone doesn't address cohesion; brushed sets must still be selected before moving. So an obvious extension to brushing is an option to automatically select a brushed set. Dynapad currently has a menu option to do so, but the relatively high cost of the operation might be improved.

But even with a convenient operation to cohere relations, a deeper problem remains: the relative structure of a diffuse set often translates poorly to another location. A translation with the intended meaning for one selected portion may have unintended results for others (Figure 3.43). There is a danger of accidental association (through adjacency or alignment) over the entire footprint of the selection. Preventing this requires careful attention to all selected items (which is even more difficult when the set is non-contiguous and non-serialized), or else moving the set to a completely clear area. Either case may demand subsequent manual consolidation.

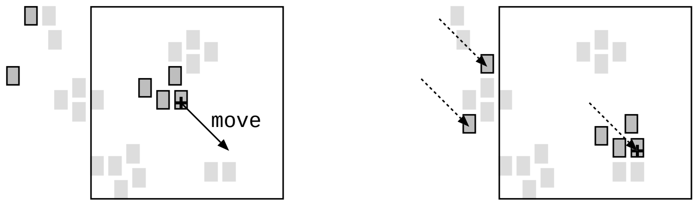


Figure 3.43: Moving a diffuse selection can include disruptive side-effects.

Therefore we can anticipate the need for an automatic *gathering* operation, not yet implemented in Dynapad. Gathering would enrich meaningful sets (perhaps already bound by a relation) with spatial continuity, which reduces the potential for accidental spatial coordination, facilitates attention, and better affords visual aggregation. Gathering would also prepare a set to be used in a contiguous region-tool, as we'll see in 3.6.

Gathering Copies. A complementary feature would be to gather not the original selected items but *copies* (which would remain tied to the originals in a brushable relation). This facilitates expressiveness: items can participate in multiple relationships, some spatially distributed, while retaining all the advantages of contiguity.

3.4.3 Labels

One of Dynapad's goals is to provide an environment which eliminates the necessity of explicit labels. Clarity is supported in other ways: the visibility, aggregation, and expressiveness of open arrangements. Nevertheless, we observed that users still choose to annotate some piles using whatever means they have, sometimes very creatively (see 4.2.1). So an obvious feature is the ability to attach explicit text labels to piles, to supplement their implicit visual identity.

The Costs of Labeling. In an effort not to bias users toward labeling everything, all objects are unlabeled by default. A new label can be created from an object's pop-up menu. Existing labels can be changed by double-clicking the text and editing with the keyboard.

Although this requires some effort, it still has a much lower cost than labeling files or piles of paper. The greater cost may be indirect: a loss of conceptual and organizational flexibility from the continual reinforcement of potentially obsolete descriptions.

"Slow-Zoom" Labels

Labels in Dynapad face a problem not shared by paper: they must be effective at arbitrary, widely-ranging zooming scales. We can presume labels play an especially important role at small scales, when surveying a collection broadly and recalling its categories. So labels' text must be large enough to read when zoomed far out. But if they zoom like other object, this would make them unnecessarily and obstructively huge at close-up scales.

Dynapad's solution is to use *slow-zooming*[22] text, which grows and shrinks more slowly than other objects. Specifically, while the world scales linearly to zoom z, labels

[22]In Dynapad's technical terms, such text is *semi-sticky* in z.

scale to $z^{0.3}$. This makes them much larger than normal when zoomed out and smaller when zoomed in. They also stop shrinking below a certain minimum size to stay legible even at extremely wide views.

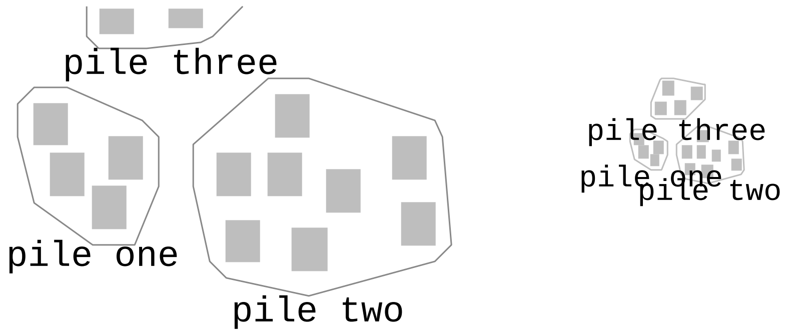


Figure 3.44: Slow-zooming labels remain at a readable size but may occlude each other when zoomed out.

A problem with this solution is that labels continually change their *footprint*, the space they occupy in the collection layout. When people micromanage a static layout, they may try to prevent occlusion, especially of text. But such control is impossible here; labels will inevitably overlap objects and each other as they grow comparatively larger in wide views.

Fortunately, this occlusion can be cleared with minimal effort and no permanent change, just by zooming. That is, zooming can serve as an *epistemic* manipulation: negotiating an effective scale to read labels.

Bound vs. Loose Labels. The slow-zooming labels described here are designed to be bound to explicit objects, such as containers. The labels follow their referents when those objects move, but the labels themselves cannot be manipulated (except for editing) — neither moved, selected, nor targeted for zooming. And just as well, for the interactive physics described above make no allowance for objects which constantly change size and occlude others.

We can imagine a variant of labels which are loose, independent objects themselves, which can be included anywhere within arrangements to annotate implicit structures. A viewer must infer their referents by proximity. However, this becomes problematic when their relative size changes drastically, interfering with whitespace and visual gestalts. Additionally, they may require their own rules of interaction.

For these technical reasons, Dynapad currently has implemented only labels bound to region-objects, which exist only in the second "layer" of functionality, described in the next section. Nevertheless, their *cognitive* role belongs to the theme of this section: supplementing digitally the expressiveness and clarity afforded by physical organizations.

3.5 Interlude: Dynapad as a Virtual Tabletop

Before introducing the next layer of Dynapad's features, in this section let's pause to consider the basic character of the collective functionality[23] described so far. In essence, this portion of Dynapad emulates digitally the affordances and costs of a paper-filled tabletop.

This first layer of features, in isolation, constitutes only a *virtual* application. It can be realized with the proper combination of libraries, but it was not particularly intended to stand alone. Nor does it correspond to any *historical moment* in development. Its more advanced features (e.g. implicit zooming cohesion, directory brushing) are newer than many of the "second-level" features of region-tools (in section 3.6). This world, as such, has never had any users.

Nevertheless, it's worth isolating as a conceptual layer whose cognitive ecology is similar to that of physical piling, considered in Chapter 2.

Furthermore, this layer it *approximates* the functionality of an earlier development phase from which we have observations of users. For clarity, we'll refer to this early version of Dynapad as "PhotoPad". Our observations of PhotoPad illuminate many important aspects of its usage, especially regarding organizational strategies, and informed the development of later features. Some of those observations have been reported elsewhere [72, 3], but this section will summarize three examples and their most important lessons.

Observational Methodology

The methodology of our PhotoPad observations closely resembles that used later in collecting the data described in Chapter 4.

[23]We exclude containers from this layer of functionality, as Figure 3.1 suggests.

We recorded video and audio data of six adult users (2 male and 4 female) organizing their photographs in Dynapad. Subjects worked with sets of their own images (ranging in size from 114 to 293), which they organized to prepare for a subsequent task, negotiated individually, such as composing a web page or scrapbook, or browsing pictures for pleasure with friends or relatives.

A Dynapad workspace was projected onto a tabletop with a ceiling-mounted video projector. The sessions were recorded using both a ceiling-mounted camera, giving a survey view of the workspace, and a camera facing the subject and operator (see below), capturing interactions between them as well as gestures produced over the workspace.[24]

We used a modified "Wizard of Oz" [16] technique: rather than operating the interface themselves, subjects were assisted by an experienced operator sitting next to them at the table. Subjects expressed their intentions by speaking and gesturing freely at the workspace, without having to learn or being constrained by Dynapad's controls. This collaborative arrangement turned the interaction with the workspace into a conversation with the operator, providing us with a running commentary on the subjects' motives and intents throughout their activity.

Environment Differences

The early PhotoPad environment in the examples ahead differs from the virtual Dynapad application described in this chapter in a few notable ways:

Photographs: Participants' collections consist not of documents but of photographs. Although this suggests no obvious cognitive implications, it seems likely that photos are more visually distinctive and identifiable than many document portraits. Therefore their effective visibility may be greater, with potentially greater clarity and stronger reminding. These are, of course, properties associated with piling.

No labels: This early version of Dynapad offers no labels. Therefore the organization must be completely implicit, another characteristic of piling.

No brushing: This early version includes no brushing, neither of common directory items nor duplicates. This limits the expressiveness of the system to relationships

[24]Some of the figures below from these observations will include subjects' gesturing hands.

encoded only by spatial placement. It also requires spatial contiguity for visual coherence.

Indirect control: Subjects do not control the interface themselves, but instead direct the operator with relatively high-level intentions (e.g. "put these here" or "zoom to those") rather than low-level input actions. Therefore we should expect the details and cost structure of the interface (e.g. the ease of selecting, dragging, and navigating) to have relatively little impact on subjects' strategies and resulting organizations. (In Chapter 4, we'll look closely at how the costs of direct control *do* influence these strategies.)

PhotoPad as a Piling Surface

The historical PhotoPad, and to some extent the current first layer of Dynapad, implement a nearly literal interpretation of a piling surface. The space itself is infinite, inert, and homogeneous, exhibiting the same behavior in all locations. There are no explicit elements that correspond to piles,[25] but clusters of items are "piled" together to represent categories, and the cognitive ecology of the environment is close to that of Chapter 2. Here we'll make a quick enumeration of the similarities and differences.

Implicit Organization: Photopad's lack of labels requires information to be expressed spatially. Even if "loose" labels (see 3.4.3) are included, much of the organization's clarity still relies on the spatial placement and visual aggregation of clusters.

Unlimited space: Zooming allows unlimited space which can be quickly and easily navigated, reducing competition for space and the need for occlusion.

Unoccluded Cohesion: Clustered items need not occlude each other; "piles" can remain "open". But at the same time, they can be moved cohesively by first selecting them. Cohesion has been de-coupled from occlusion.

Multiple Classification: Because the digital collection elements can be copied much more easily than paper documents, they can more easily be multiply classified.

[25] Paper piles are themselves only aggregates of their elements.

This can be even more effective in the full-feature environment, when copies are linked by brushing.

Exposure: Items are exposed; their easy visibility and accessibility facilitates piling. But there is an important difference between their exposure and that of paper piles in an office: Dynapad is not the user's primary work environment, sharing its workspace with unrelated activities. Dynapad is an insulated environment for the specialized activity of organizing collections.

In that respect, all the effects of visibility and accessibility occur only in the context of that activity. Workspace clutter will not offend in daily life, for example, and piles are not disrupted by reusing desktop space for mail-reading. But relative to the outer world, everything in Dynapad is less visible and accessible than a paper pile on one's desk.[26]

3.5.1 PhotoPad Examples

With that cognitive ecology in mind, let's examine three examples from our PhotoPad sessions (also summarized elsewhere [72, 3]) which illustrate some important organizational strategies that emerge in that world.

Example 1: Multi-scale Expressiveness

Figure 3.45 shows one participant's collection, photos from a vacation to Hawaii, invested with multiple levels of spatial structure. At the highest level, the space is divided into upper and lower halves, and each is organized differently. The lower half is categorized by subject (e.g. "sunsets" or "flowers"). The upper half consists of event-specific piles arranged in chronological order (left to right, top to bottom). One of these, "the volcano" (circled and enlarged), is a cluster of three subpiles: two panoramas and a series of landscapes arranged by location, color, and saturation.

Furthermore, according to this subject, the landscape sub-pile was even shaped like a volcano: wide at the bottom and narrow at the top. The aggregate shape of the pile served as an additional reminder of its meaning, an example of aggregation

[26]For that reason, it seems more appropriate to portray Dynapad as a "tabletop" than a "desktop", which invokes the multiple purposes of an operating system's generic environment.

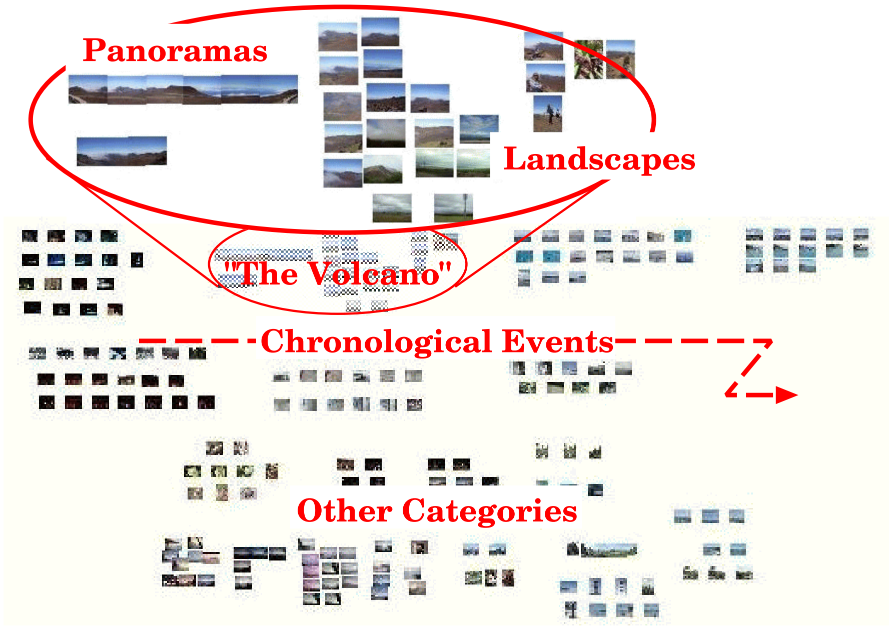


Figure 3.45: A richly-structured, multi-level workspace with sorted piles

enhancing expressiveness (see Figure 2.25). However, such expressiveness also makes the pile non-serial, unlike most of the other event piles – an example of the tradeoff between expressiveness and seriality (see Figure 2.24).

This complex organizational scheme was not anticipated by the subject at the start, nor was it ever explicitly articulated. It emerged gradually through reflective interaction with the collection.

This workspace demonstrates the following important themes:

Heterogeneous & Nested Organization: Different organizational schemes are used at different levels and in different areas within a level. Space itself is uniform, but the *use* of space is not.

Opportunistic Expressiveness: Organization schemes are possible which are highly customized to the particular content of the collection. The most expressive arrangements probably cannot be anticipated in isolation from content, prior to the user's cycle of reflection.

Organization by Time: Some arrangements, however, are likely to recur: organizing by time, in particular, has been shown to be a ubiquitous and effective strategy not only for photographs [30, 31, 39] but for personal media more generally [20, 22, 28, 54]. The identifiable time course of the events in Figure 3.45 provides a structure which influences the entire workspace.

Interestingly, the availability of this structure was largely an accident of design. All the imported photos came from a single directory and bore their default filenames (e.g. "DSC00123.JPG"), assigned sequentially over time. Dynapad's initial layout grid sorted them by filename, *coincidentally* sorting them by time as well. In the resulting grid, photos from the same event were adjacent and events were in relative order. Representing this organization involved mostly extracting partial rows of the grid and separating them into piles. This also explains the tidy alignment of the rows in each event pile, inherited from the initial grid.

The next example shows some consequences of a less fortuitous initial layout.

Example 2: Micro-scale Enrichment

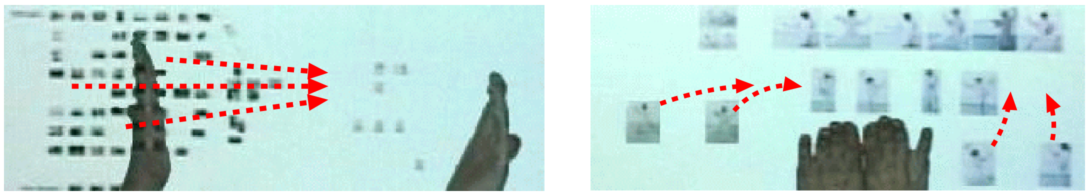
Another subject had already invested some effort in organizing her photos into directories and naming them informatively. Her workspace's initial layout was therefore a set of directory-grids (in an "outline" format, as in Figure 3.40(b)), and the photos within them, sorted by name, were in effectively random order.

For this example, we'll focus on an action with two phases. The first phase selected (incrementally) a dispersed set of related photos from the initial grids and extracted them toward the right (Figure 3.46(a)). This left-right separation was dictated by the vertical layout of the outline.

The second phase consolidated the extracted photos, adjoining and aligning them into two rows for easier comparison. The subject not only made a pile with the category, but made a deliberate investment in its microstructure, its *seriality* and *contiguity* – properties that paper piles have naturally. Ironically, some of that enrichment might have emerged automatically in the initial layout, as with the earlier example, if not for the subject's prior investment in labeling the files.

This episode illustrates these themes:

Drawing from a Reservoir: A basic cycle of activity shared by all participants is to


(a) Extracting items from the initial reservoir. The hands' gesture means "bring them this way".

(b) Adjoining and aligning those selections. The gesture means "adjacent, like this".

Figure 3.46: Extracting and enriching a category.

repeatedly draw selections from an initial "reservoir" and categorize them, gradually adjusting and arranging those categories. The reservoir becomes a space with a specialized role, and it shapes the eventual structure of the workspace.

Enriching Microstructure: Regardless of the relative positions of piles, their internal structure is consistently enriched with seriality and normalized spacing.

Example 3: Negotiated & Overlapping Usage

Our final example shows another participant in mid-session (Figure 3.47). He has pulled out multiple sets of photos from the initial layout grid (within the marked rectangle) and placed them in piles (circled) wherever empty space was available, including the "holes" left from previous selections. Not only do different areas begin to play different roles, but they sometimes overlap.

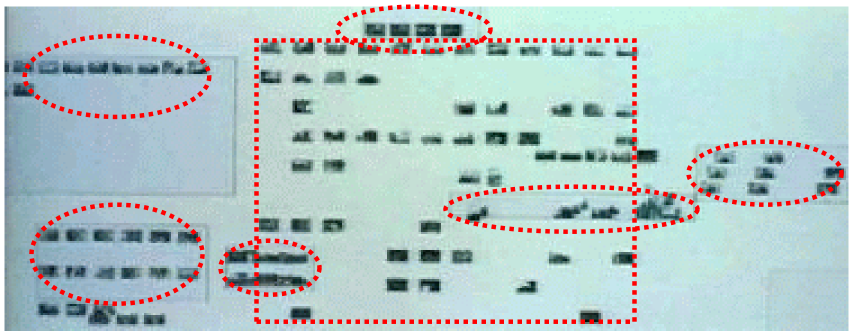


Figure 3.47: Overlapping spatial roles

In this case, the piles are explicitly *insulated* from the reservoir in containers (see 3.2.4). It's unclear to what extent this explicit demarcation is required for overlap, but it surely contributes. This subject chose not to take advantage of Dynapad's unlimited space, perhaps because the stability of the contained piles reduced the need for clean separation. In terms of the ecology of Chapter 2: available space wasn't needed to improve stability because artificial cohesion was sufficient.

Negotiated Use of Space: The use of space is negotiated in a series of steps, each driven by the accumulation of previous choices. The structure of a workspace at the end is determined by opportunistic, moment-to-moment revisions of the landscape of the developing space.

Cohesion Improves Stability: As with paper piles, cohesive and explicit structures can overlap with other structures without disruption.

3.5.2 Summary: The Variegated Use of Space

Past research, notably [42], has illuminated many ways in which spatial organization is used to facilitate cognitive activity. Many of these techniques are relatively transitory, supporting specific phases of processes. Similar dynamics emerge in Dynapad, and Chapter 4 will consider these in more detail.

But for now, as a summary of the examples above, let's take an essentially atemporal view and consider how space is organized in an arbitrary "snapshot" of a PhotoPad workspace. What we need now is merely the recognition that *space is used differently from one place to another*. We can identify at least three types of spatial variegation, which correspond roughly to three different scales:

Varied Workflow Roles At the largest scale, across the entire workspace, different areas play different *roles* in the workflow of organizing a collection. The simplest examples include two roles: a source *reservoir* and the destination *arrangement*, where expressive structures encode categories and other relationships. Additional roles include intermediate staging areas, where items might be consolidated before final placement.

Other work has shown similar functional specialization in different areas of digital environments [51, 52].

Varied Meaning At intermediate scales, different areas within arrangements have different *meanings*. Items are categorized and essentially annotated by their placement, and the interpretation scheme varies in different areas. Arrangements often include empty space to separate categories and changes in organization. This gives such areas distinctive “textures” or “geographies”. The collector’s familiarity with that geography, mediated by the clarity of the organization, affords relatively direct access to particular categories.

Internal Consolidation & Enrichment At the smallest scale, the area *within* a category is often enriched with seriality, contiguity, and alignment. These areas correspond to the internal structure of piles.

In the lowest-level categories with little internal organization, individual items are found primarily by serial (or random) search, for which a contiguous and serialized arrangement is ideal. Contiguity also decreases the cost of emulating cohesion by selection.

Of course, these qualities may not be nested in the order above. A densely consolidated pile, for example, may contain subcategories or even role-specific internal zones.

3.6 Region Tools

The *physics* of a workspace are simply the ways objects behave and the affordances they offer [9]. In that sense, this entire chapter has been about the physics of Dynapad. So far those laws, like those of the physical world, have been spatially uniform: the appearance and behavior of objects are the same everywhere.[27] Although the *use* of space is variegated, so far space itself has been uniform and inert, as on a tabletop.

[27]One exception is the differential behavior inside and outside a container, a preview of the *regionalized* physics considered in this section.

Variegated Physics. An environment's physics both support and perturb participants' behavior. As suggested in Chapter 1, an ideal physics is one which guides the user into behavior which proves effective in her broader practice (collection management, in this case). But for any particular region of space, its ideal physics depend on the cognitive role that space plays. Since the role of space varies by location, ideally the physics of space should be similarly localized.

Specifically, drawing on our earlier observations of PhotoPad workspaces, this means that Dynapad's physics should support, at three different levels:

- variation in the *workflow roles* of different areas;
- variation in the expressible *meaning* of different areas of organization (including piles) and the relations between them;
- *enrichment* of space within piles to facilitate serialization, contiguity, cohesion, de-occlusion, and other properties considered in Chapter 2.

Keeping Physics Humble. But as we've also seen, these uses of space are often *intermingled*, *overlapping*, *opportunistic*, and *negotiated* in a gradual cycle of reflection and revision. The meaning and role of a space are changing and portable.

The environment designer's challenge is to *provide* but not *prescribe* automation; we want enough but not too much. We want to harvest and illuminate legitimate structure but not, as with Pandora's shoebox, to fabricate it. To remain exploratory, automation should be reversible and repeatable. In short, physics should be *humble*.

Localizing Physics with Region Tools. This section introduces a new layer of functionality in Dynapad: *region tools*. We've already seen one simple example, the *container* described in section 3.2.4. Later we'll situate containers in a larger taxonomy of tools, but their primary function is to enrich a space with cohesion, making it portable.

Region tools offer a generalized *protocol* by which a user can wield local and humble physics, specifying what to automate where, much like the operational syntax of individual actions.

Some Related Work. Similar localized tools have been developed in others' work. In particular, Bier et al.'s *Toolglass* and *Magic Lenses* [12] offered compelling early examples of customized computation applied to spatial regions. Similarly, Kang's "semantic regions" [40, 41] apply various spatial layouts, representing different mental models, to personal media objects that are dragged onto them. In another example, Scott and Carpendale and colleagues have utilized portable "storage bins" [64] and other territory-specific automation [65] in collaborative tabletop envionments. And again, Dourish et al.'s *Presto* system [19] includes areas where automated document retrieval can be customized by adding or removing exemplars.

3.6.1 Clumps

Let's begin the description of Dynapad's regional tools with the simplest, most literal interpretation of a pile: the self-adjusting "clumps"[28] of Figure 3.48.

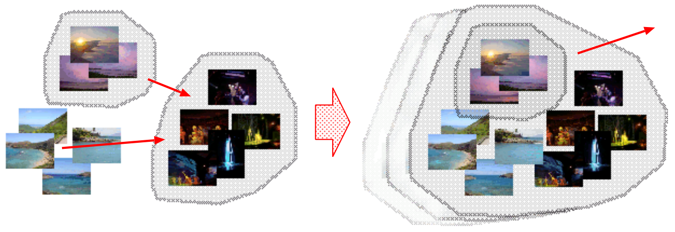


Figure 3.48: Dynapad's self-adjusting, portable "clumps"

These clumps are a type of container; that is, they are portable and carry along any portraits or containers placed inside them. They also have an unambiguous boundary which clearly demarcates its contents. In this way, clumps reify implicit groups into explicit units which offer cohesion without prior selection.

Also, like other containers, clumps can be made to solidify and evaporate at very small and large scales, as described in 3.2.4.

Furthermore, a clump automatically adjusts its border when members are added, moved, or removed. Specifically, it maintains a convex hull (a "rubber band") around its members, padded with a margin proportional to their average diameter. This

[28] Although these objects are Dynapad's most direct analogue of physical piles, we call them "clumps" to leave the more general term "pile" for the broader class of aggregate structures, including other region-tools.

margin ensures that there is always room to add another similar-sized member at the clump's edge. The margin also keeps an emptied clump from collapsing completely so that new members can be added.

Exposure, Cohesion, and Insulation

Apple's piles [50] require some form of "browsing" to expose and interact with piled elements. As discussed in Chapter 2, this default lack of exposure reduces visibility and accessibility and their concomitant advantages. Unstacking these piles would normally gain continuous exposure by giving up cohesion. Selection of entire groups can synthesize cohesion but requires some effort and risks errors. But Dynapad's clumps and other containers offer both continuous exposure and effortless cohesion.

Clumps also offer better *insulation* than loose objects: not only do intended groups cohere together, but nearby items which are *not* members are clearly excluded from the clump. Cohesion and insulation play complementary roles in improving organizational stability: cohesion keeps members in, and insulation keeps non-members out.

Implicit vs. Explicit Regions

An additional benefit of containers is that their explicit boundary establishes a condition for further automation, or *regional physics*.

In Dynapad's basic "tabletop" functionality (e.g. Figures 3.45–3.47), the "piles" are *implicit*: they exist only through the relative spatial density of their elements, and their meaning is only in the mind of the participant. But region-tools like clumps are *explicit*: the spatial structures are reified as tangible, portable objects with definite boundaries. Our goal is to provide "proactive" piles, whose behavior is a function of the pile's *meaning* or *role* within the workspace. One might imagine that such piles could be implemented equally well as explicit tools (as in Figure 3.48) or implicit regions (perhaps with a clustering algorithm). But such implicit piles are problematic because their "meaning" cannot "track" the pile if it moves.

An example of the problem is illustrated in Figure 3.49. Initially, group A occupies space S. The subject first moves group A downward, then moves group B into

A's former space (S). Should the "meaning" of pile A follow the objects, or remain in the space and apply to the new group B? Put another way: did pile S *move* or *change contents* (replacing A with B)? We cannot correctly interpret the action without knowing the user's intent.

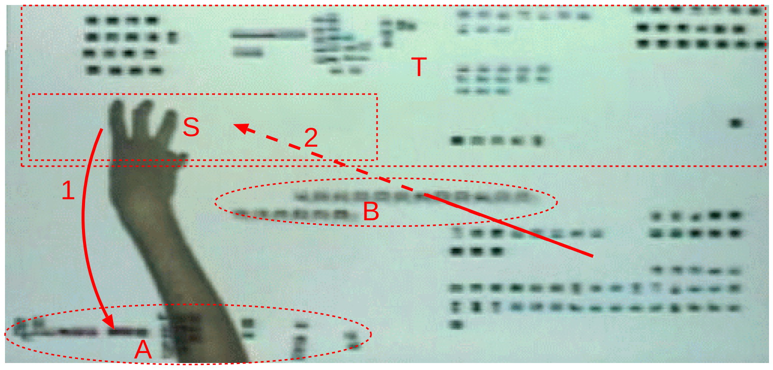


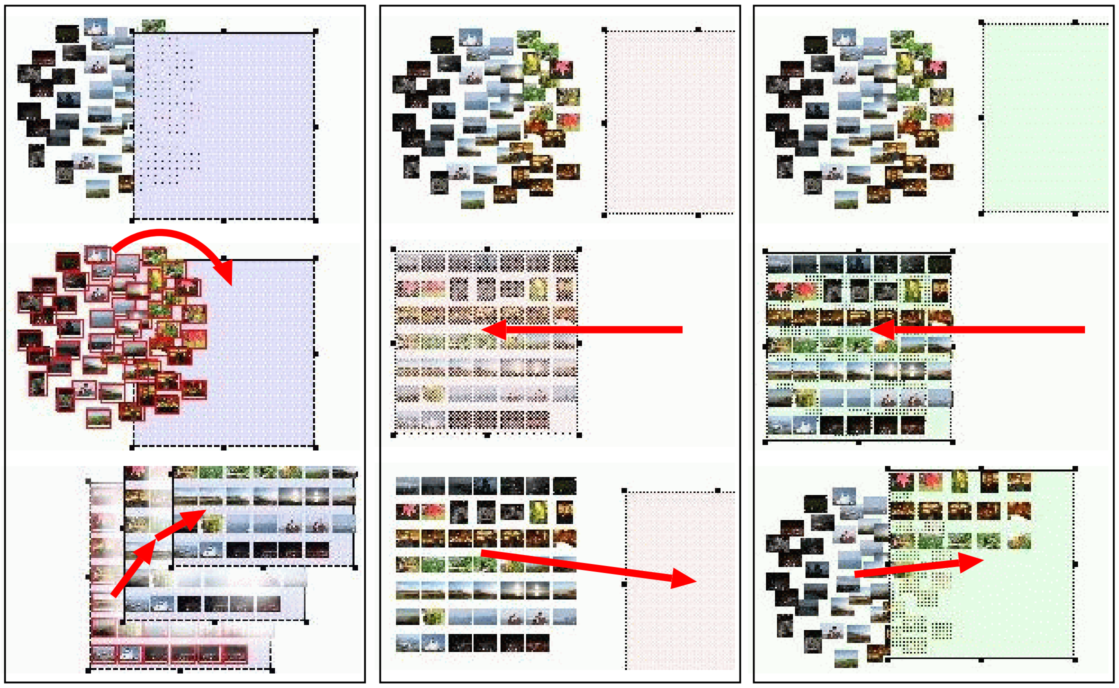
Figure 3.49: Ambiguous move of implicit piles

From the subject's self-reporting, we do know the correct interpretation: he has decided to use the top of the workspace (area T) as a timeline, and has arranged chronologically several event-specific piles. Group A, composed of "sunsets", was placed earlier and does not follow this scheme. Therefore he moves A out of the way to make room in the timeline for a new event, B. The former meaning ("sunsets") of space S *moves with the pile*, and that space is reclaimed by pile T to include sub-pile B, with its own meaning.

If the piles are *explicit*, like those of Figure 3.48, where the pile's behavior is bound to a tangible object, the user's decision to move the *pile itself* or just its contents disambiguates the interpretation. For this reason, although implicit versions are possible, all of Dynapad's region-tools are explicit, draggable objects. They reify meaningful groups of elements into explicit structures, remain open and accessible at multiple scales, are intuitive and nearly effortless to modify, and yet protect any substructure the user invests in them when they are moved, abutted, or even superimposed.

3.6.2 Arrangement Tools: Tray, Stamp, and Lens

Despite their useful structure-preserving affordances, the clumps of Figure 3.48 are a narrowly literal interpretation of the "pile as specialized region" metaphor; besides adjusting their boundaries as elements are added and deleted, they offer no other *proactive* behaviors to assist the user in *creating* meaningful structure. So, our next step is to enrich piles with *automated layout*.



(a) Tray (b) Stamp (c) Lens

Figure 3.50: A sample of grid-arrangement tools

Figure 3.50 illustrates the behavior of three such automated region-tools: a *tray*, a *stamp*,[29] and a *lens*. Let's consider first what they have in common: each tool arranges a set of collection elements (photos in this case) into a grid, which serializes them. At the same time, it sorts them by several possible criteria; here, the photos are sorted by the date and time they were taken.[30] This "gridding" and sorting automatically enriches each

[29]In past work, we've also referred to the stamp tool as a "mutator" or "magnet".

[30]The date information is extracted from the *exif* header embedded automatically in the image data

arrangement with a microstructure much like that invested manually in Figure 3.46(b): similar photos are aligned and adjoined (by virtue of their similar timestamps), thereby aiding comparison. In other words, each arrangement is given seriality, contiguity, and order, as Figure 3.51 suggests. And these in turn have the potential to help guide attention.

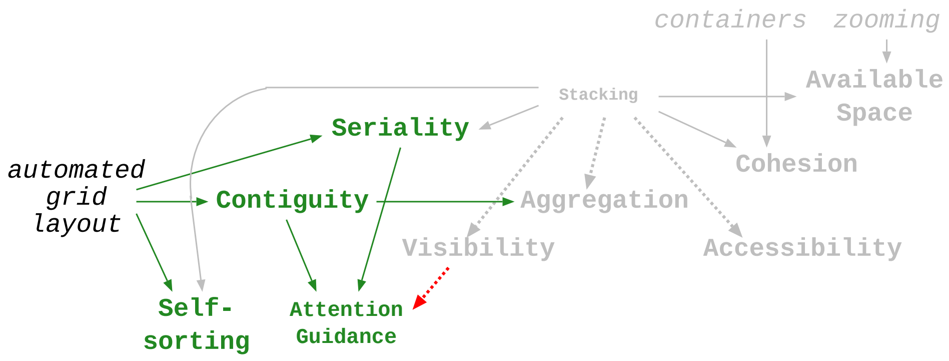


Figure 3.51: Grid arrangements improve seriality, contiguity, and sorting

So how do these three tools differ? The *tray*, like the clump-tools above, is a container which carries around (and rearranges) objects dropped into it. The *stamp*, in contrast, applies its effect wherever it's dropped, rearranging objects within its boundary but leaving them behind when moved. Finally, the *lens* leaves undisturbed the objects it's placed over, but instead arranges "phantom" copies of them projected above it.

Dynapad's system of region-tools is designed to be flexible and extensible, so its architecture separates different aspects of the tools' behavior into independent components which can be combined in multiple ways. Although there are many alternatives to the current architecture, we believe that the separation we have adopted reflects important design dimensions. These two basic categories of the tools' features — what they share, and how they differ — reflect an important distinction, which we'll discuss next.

by the camera.

3.6.3 Generalizing Tools' Effects

One component of a tool's behavior is its *effect*, what it does with the objects it acquires. The three examples above share the same effect component, ordering their contents by time in a grid. But we can imagine countless other operations on sets of images. An effect may impact both how an object is *positioned* (as with these examples) and how it is *displayed*. Many classical implementations of lenses [12, 25] are display-only: for example, highlighting objects of certain parameters, displaying labels, making corrections to text, and so on. We have only begun to explore the possibilities of displaying metadata and otherwise informative depictions of papers and other digital content. However, because of people's extensive use of space in organizing information, and because local spatial rearrangement presents unique challenges for lens-like tools, we have focused especially on arrangement effects like those in Figure 3.50.

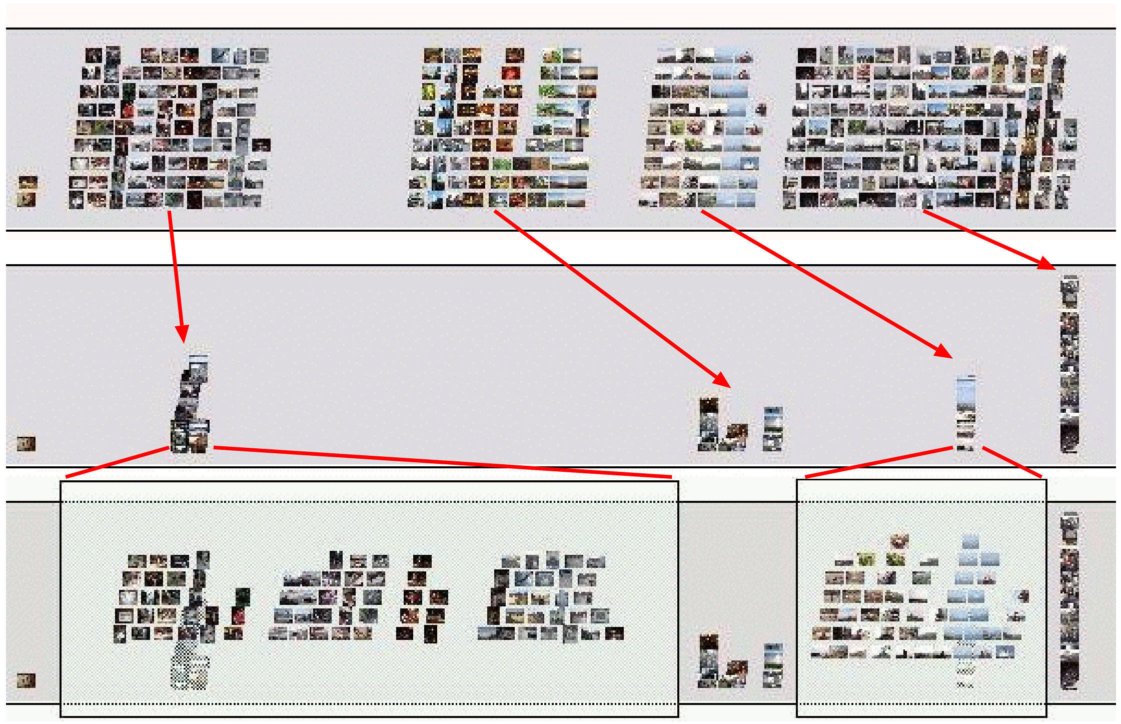


Figure 3.52: Timeline effects, top to bottom: Loosely-clumped, Precisely-spaced, Precise with additional timeline-lenses.

A variant on grid-arrangement is a *timeline*. As illustrated in Figure 3.52, the timeline spacing can be adjusted to spread out objects for greater visibility (top panel) or align them precisely according to their timestamp, producing a histogram effect (middle panel). The resulting columns are sloped to the right since each item's horizontal position reflects its exact time; the vertical stacking ascends only far enough to clear room at the "ground" level for the next item. Therefore the height of columns reflects the density of items at that time.

The bottom panel of Figure 3.52 shows how tools can be combined: a tightly-clumped timeline tray is overlaid with two additional lenses to "pull apart" two of its clusters and show their temporal substructure. This creates in effect a non-linear magnification across the timeline, a "fisheye" view with multiple foci.

Even more than the time-sorted grids, the timeline helps to categorize photos *by event*, a strategy employed frequently by our subjects (as in Figure 3.45, for example).

Both the timeline- and grid-arrangement effects automate several common arrangement subtasks (sorting, adjoining, and aligning) and the timeline also automates spacing. But these tools' ease of use costs them *expressiveness*: they cannot be invested manually with the same richly-detailed substructure as basic clumps.

As a compromise, we have anticipated developing another layout variant: the "gathering" clump. Basic clumps do no arrangement and force the user to manually adjoin, align, and de-occlude items. A "gathering" version (Figure 3.53) would draw items into a tight grid or other pattern, like marbles in a rubber-band, providing automatic contiguity and seriality without imposing any particular sorting.

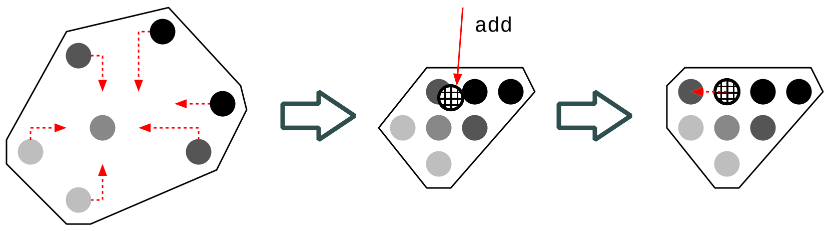


Figure 3.53: Proposed "gathering" clump

Another potential benefit of a gathering-clump would be to consolidate a distributed group (for example, a brushed set). For this purpose, a gathering-*lens*, which consolidates *duplicates* of objects, might be best.

Metadata Parameters. The arrangement examples so far have all used a timestamp, extracted from the *exif* data automatically encoded in an image by the camera. However, this timestamp is but one example of a *metadata parameter*: a function extracting some value from an object for use in various effects (like layout). For photos, other *exif*-based parameters might use camera focal length, for example, to distinguish close-ups from landscapes, or brightness to distinguish indoor and outdoor scenes. In general, parameters can utilize any source of "faceted metadata" [74] for photos or other content, and can be combined with existing effects — for example, a timeline can easily become a "nameline" where items are sorted and spaced alphabetically by filename or owner.

Tools' Operational Syntax

The differences between the tray, stamp, and lens examples illustrate another important design dimension. Although the tools have the same arranging effect, they differ in the rules by which they acquire and retain their contents. If we think of a tool's effect as an operation, those rules of acquisition determine the tool's operational syntax (see 3.2.2).

The syntactic rules for region tools involve both spatial and temporal relations between the tool and its operands. For implicit piles, the relation between a region and its members is merely spatial: a pile comprises those objects within its implied spatial boundary. However, when the pile is an explicit, movable object, its relation to its contents requires a more precise operational definition. The *history* of an arrangement may be important: for example, there may be a difference between moving an object into a region and moving a region over an object, or between moving items one at a time or as a batch.

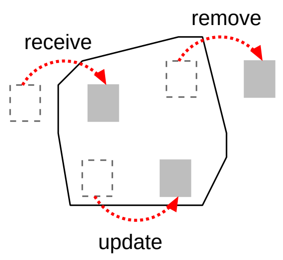
While a tool's function (its effect) determines the conditions and organizational strategies in which it is useful, it is a tool's syntax which determines its *affordances*, how you *invoke* its function, how you *interact* with it during use. We have implemented these different variants of syntax not to search the design space for a single, consistently "best" design, but because we have observed these variants produce qualitatively different usage strategies. We'll consider such differences later in Chapter 4.

3.6.4 Implementation Details

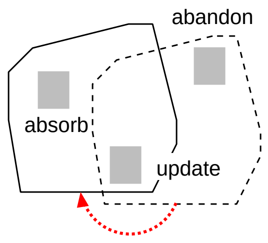
All of Dynapad's region tools share a common architecture based on a simple principle of "membership updates": regions act when objects enter, leave, or move within them.

Each region *effect* (e.g. timeline-arrangement) provides an interface invoked by callbacks on four different events:

- A moving region can *absorb* new members and *abandon* current members. Note that these account for cohesion and insulation: non-absorptive regions are insulating, and non-abandoning regions are cohesive.
- A stationary region can *receive* objects moved to it or *release* members moved out of it.
- Either can *update* members which move within it.
- Finally, a region can perform a *finish* action once all membership changes have been made.



(a) Moving objects, stationary region



(b) Moving region, stationary objects

Figure 3.54: Callback events for region-tools

Although membership is discrete (in or out), mathematically continuous effects (e.g. a Gaussian lens) can be implemented by considering a member's exact position after an *update* and extending the boundary of the region to include the entire domain where the effect function is above some threshold.

A region's *syntax* specifies which of the events are active — for example, a tray does not *absorb* or *abandon* its members (so it is cohesive and insulating). Conversely, a lens both absorbs and abandons the objects it overlaps but not the "phantom" duplicates it carries. Syntax also determines several other policies for maintaining region membership. We have found that a wide range of syntax variants may be specified with only five binary variables:

- An *absorptive* region may *absorb* and *abandon* objects.
- A *proxy* region (a lens, for example) applies all effects to copies of its members rather than the originals. This ensures that it does not disrupt anything in the workspace or other regions, even if they share its members.
- A *greedy* region *receives* first and tries to receive all dropped objects (even if it does not spatially contain them) whenever it encloses the drop location (i.e. the cursor position). This affords "funneling" a large selection into a small container or button; it is an intuitive default in many applications.
- A *retentive* region carries along all of its members (or proxies, if a lens) when moved.
- A *possessive* region does not share its members with any other possessive region; that is, objects cannot belong to more than one possessive region.

These properties of our three syntax examples are summarized in Table 3.2.

Table 3.2: Summary of region-tool syntax variables

Property	Tray	Lens	Stamp
Absorptive (acquires underlying objects)		X	X
Proxy (operates on copies)		X	
Greedy (takes all dropped objects)	X		
Retentive (carries members)	X	X	
Possessive (limits sharing)	X		

Algorithm

Dynapad's actual membership-update algorithm is complex due to optimizations and implementation-specific details, but its worst-case scenario[31] can be expressed

[31]The complexity of the worst case is $O(n \times r)$, where n is the total number of objects (including regions) and $r \leq n$ is the total number of regions.

relatively simply. Whenever a set of moved objects Ω_m, which may include regions, are moved and "dropped" with the pointer at point P:

1. Find "damaged" area D, the spatial union of Ω_m before and after the move.

2. Find all objects ($\Omega \supset \Omega_m$) intersecting D, all regions ($R \subset \Omega$) intersecting D, and moved regions ($R_m = R \cap \Omega_m$).

3. *Moving objects:*
   For each region $r \in R$ (ordered top-down)[32] and each moved object $\omega \in \Omega_m$, $\omega \neq r$:

   (a) if r uses *proxies*, ω_r is a private copy of ω, else $\omega_r = \omega$;

   (b) $\omega_r \in r$ if r *contains*[33] ω,[34] or
   r *contains* P and r is *greedy*;

   (c) if $\omega_r \in r$ now but not before, r *receives* ω_r;
   if $\omega_r \in r$ both before and now, r *updates* ω_r;
   if $\omega_r \in r$ before but not now, r *releases* ω_r.

4. *Moving regions:*
   For each moved region $r \in R_m$ (ordered bottom-up) and each object $\omega \in \Omega$, $\omega \neq r$:

   (a) identify ω_r as before;

   (b) $\omega_r \in r$ if r *contains* ω;

   (c) if $\omega_r \in r$ now but not before, r *absorbs* ω_r;
   if $\omega_r \in r$ both before and now, r *updates* ω_r;
   if $\omega_r \in r$ before but not now, r *abandons* ω_r.

5. For each $r \in R$, r *finishes.*

[32]Emulating gravity, a moved ω is *received* from the top and a stationary ω is *absorbed* from the bottom.

[33]In our current design, r *contains* ω if r encloses ω's center and ω does not enclose r.

[34]For lenses to be compositional, the output ω_r of one must be the input ω of another. Currently we do not permit this. To do so would require that any proxy ω_r is incrementally added to $\Omega - \Omega_m$ prior to step 4 above. For an exhaustive discussion of lens-composition challenges, see [25].

3.6.5 Summary of Tools

The design space of Dynapad's region-tools can be characterized in a relatively minimal interface of four callback events and five binary properties, combinations of which correspond to several intuitive interaction metaphors. There is a natural and productive dissociation between two facets of a region-tool's behavior: its *effect*, the operations that it performs on digital objects, and its *operational syntax*, how we interact with it.

The arrangement trays, lenses, and stamps described in this section are but a few examples of the vast space of possibilities for region-tools. We emphasize that because the different components of the tools' behavior are independent and complementary, any new effect, parameter, or syntax created can be combined with existing components, thereby creating many tools with relatively little effort (for example, a "filename-grid-lens", "filename-line-tray", etc).

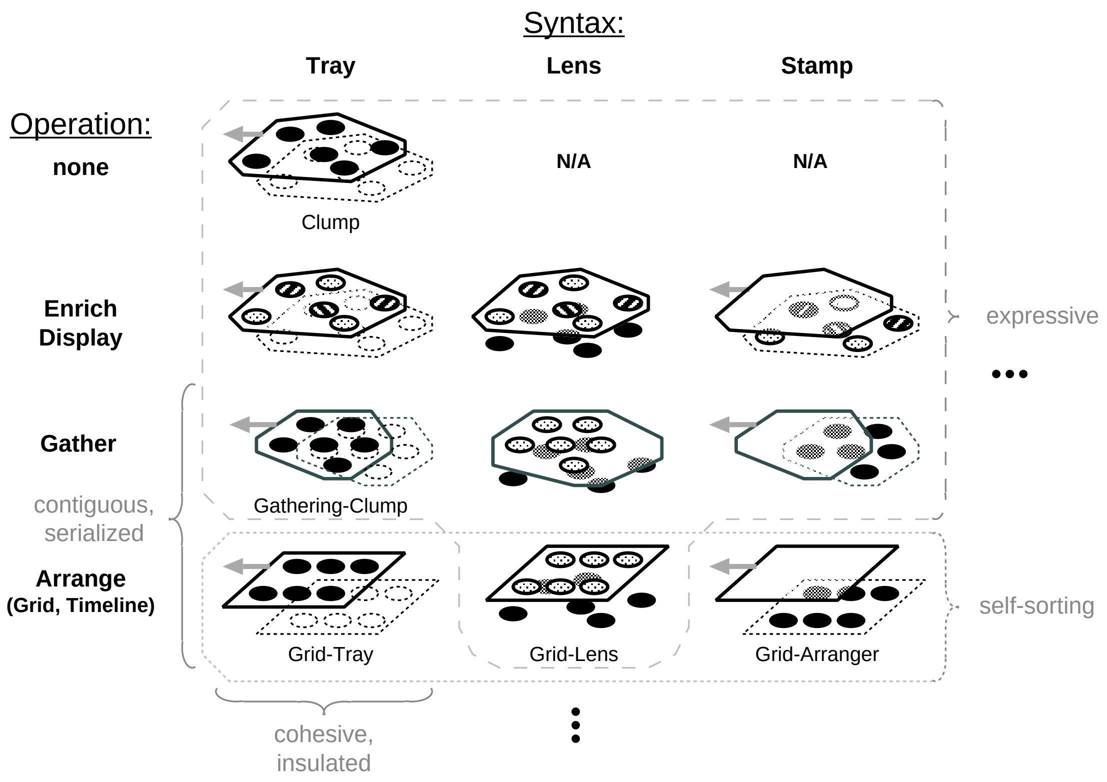


Figure 3.55: A small sample of the design space, including the tools detailed in this section.

Figure 3.55 illustrates a small sample of that design space, including the examples we've discussed. The basic clump has essentially no effect beyond the cohesion resulting from its tray-like syntax. Traditional lenses and see-through [12] or spatially-situated tools [55] are included in the space as two forms (lens, stamp) of a display-only effect.

These are all "piles", in a very loose sense. They are all derived from the same basic cost structure, described in Chapter 2. But these designs have begun to stretch that metaphor. Their first adaptation is to unstack themselves, increasing their exposure. From there, they selectively *synthesize* various affordances they would lack in the physical world. They vary in their individual affordances and tradeoffs, gradually bridging a continuum between literal stacks and versatile regions with customizable, non-intrusive, local physics appropriate to their roles in the organization.

3.7 Generalized Interactive History

In all activities in which we interact with physical or virtual materials, two things are produced at the same time: an *artifact* (i.e. changes in the world) and an *experience* (changes in our head). In Dynapad specifically, the artifact is the organizational structure we gradually imbue into our virtual collection. The experience comprises the multitude of small decisions and discoveries we make along the way; many these leave their mark in the artifact, but all of them revise in some way our *conception* of the collection and its members. This process of sensemaking is tightly coupled to the history of the artifact. So we can potentially improve our understanding if we can preserve and reflect on that history. Such is the motivation for many forms of history-enriched digital objects [34].

The third layer of Dynapad's functionality is a mechanism for revisiting the history of a workspace and the activity which shaped it. The goals of this section are threefold. First, like the previous sections of this chapter, I'll describe in detail Dynapad's existing functionality and its design rationale. Alongside that description, I'll identify some principles that *any* such system must satisfy. Finally, this discussion will establish the necessary background for Chapter 4, which describes how Dynapad's history mechanism is adapted as an analysis tool to supplement our video data.

Related Work on Interactive History

Previous research has explored many ways of recording and utilizing activity histories. This includes physical, real-world activity, which can be captured on video. Considerable research to this end has developed systems for exhaustively recording activity (e.g. MyLifeBits [28]) and facilitating video analysis [53], automatic indexing and summarization [1] or "gisting" [13], often for the purpose of reflecting retroactively on one's own activity. Another purpose, of course, is to enable a researcher or analyst to study the participant's activity. This work too includes analysis of video data (see Chapter 4).

Other research has explored the use of one's personal history as a context for organizing and retrieving digital information (e.g. *Stuff I've Seen* [20], *TimeScape* [54], and *LifeStreams* [22]).

However, let's restrict the immediate discussion to interactive software *environments*, like Dynapad, whose primary purpose is to support a particular activity and offer some access to history within that environment as a supplemental feature.

Consider the common web browser, an environment for *navigation.* A browser offers some access to history by keeping logs of recently visited sites, and a *stack* for the most immediate. Certainly users benefit from this ability to retrace their steps. Both of these formats are linear, however, and make poor allowance for *branching* after backtracking. More sophisticated representations may retain the branching structure and possibly supplement it with thumbnails of each view in the history [33].

Goals of a Historical System

With these precedents in mind, let's summarize the core affordances that a history "playback" can provide:

Visitation: revisiting specific earlier views of the artifact, to see again exactly what was seen before. The purpose might be, for example, to validate a memory, or to recall and reassess a specific decision. Since we cannot know in advance which moments will be important, ideally the history should be *exhaustive*, offering a snapshot of the artifact at every state.

Context: Sometimes the *flow* of history is more important than the details of any particular moment. So individual snapshots, even if exhaustive, may be inadequate unless they are *contextualized* with connections to each other. The system should be able to infer and represent *processes*, not merely states. This may require that it be "instrumented" to record *events* and not merely their effects.

Restoration: The basic practice of saving and reloading files is a form of historical "playback". For this purpose, the critical property of saved versions is that they are not merely viewable, like a screenshot, but also *functional*: they can be fully reactivated in the original environment as if without interruption.

A saved state which is functional is necessarily also visitable. So an *exhaustively functional* history, in which all past states are restorable, subsumes the affordances of visitation above, and offers additional benefits.

Most obviously, an exhaustively functional history provides a robust automatic backup — a good idea for any application, but particularly so for an evolving and unstable prototype like Dynapad into which real users invest real effort that must not be lost. Indeed, the auto-backup aspect of Dynapad's history has recovered more than one crashed data-collection session.

But another advantage is that when users have confidence that all states are recoverable, they can be more willing to *explore*, to try manipulations and "destructive" tools whose effect may be unpredictable. By reducing the cost of saving (to zero!), Dynapad potentially increases flexibility: users can try out new arrangements without losing the old ones.

So, in pursuit of these goals, Dynapad's history mechanism is exhaustively functional and connective. It automatically saves a user's every[35] action, can reconstruct any past state, and can replay or rewind along any thread of history.

3.7.1 An Abstract Model

Dynapad's implementation is of course only one possibility of many which would satisfy the goals above. So before examining the details of Dynapad's particular design,

[35]The mechanism is universal but not yet triggered for certain actions; see 4.1.4 for details.

let's consider an abstraction of the system which illustrates some inevitable constraints on the design.

Connecting Heredity vs. Chronology

Consider the following simple scenario in a workspace initially in some state **a**. Any changes produce new states, e.g. **b** and **c**. Then *restore* **b**, either by "loading" it or "undoing" the last action. Finally, make a different change, producing state **d**.

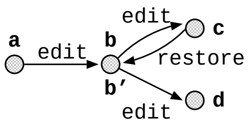
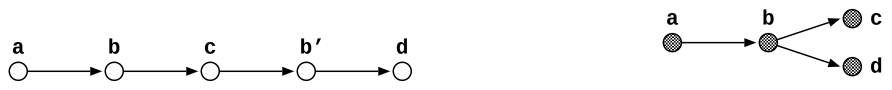


Figure 3.56: A backtracking sequence of actions

This example illustrates a crucial difference between two types of historical context: chronology and heredity. The workspace's *chronology* is the linear sequence of episodes in the user's experience. The moment revisiting **b** (which we might call **b'**) is distinct from **b** itself. Both are *views* of the same state.

In contrast, the workspace's *heredity* is a branching "tree" of states representing the developmental history of the artifact. Each state is derived from one "parent" state, from which it inherits structural similarity, even if unrelated visits elsewhere (e.g. **c**) intervene in their chronology.



(a) Chronology, a series of "views" (b) Heredity, a tree of derived "states"

Figure 3.57: Two aspects of historical connectivity

Heredity's branching structure in an inevitable consequence of exhaustive functionality. If past states remain functional, new interactions with them will produce new states. And if all moments in the chronology are to be preserved, both the old and new branches must coexist. Regardless of its implementation, *any* exhaustively functional history must have non-linear connectivity.

A data model which makes the necessary distinctions between states, views, and their connections is equivalent to a two-part graph, as in Figure 3.58. States are linked "one-to-many" to each other by inheritance. Views are linked serially (chronology) to each other and "many-to-one" to their corresponding states. For convenience, we can name corresponding views and states to associate them (e.g. state **b**, views **b** and **b'**).

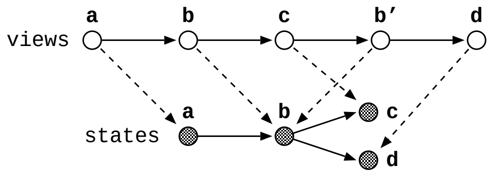


Figure 3.58: The core data model of interactive history

Connective Structure in Related Work. Most applications which preserve history are not exhaustive, throwing away all branches but the "current" one. An "undo/redo" stack in editors and the "back/forward" stack in browsers are typical examples.

Other applications' histories are non-functional; they allow no branching to begin because past states are inactive. Examples include Rekimoto's *TimeScape* [54] and Fertig et al.'s *Lifestreams* [22]. Such applications' primary concern with history is to preserve the *chronology* of the user's experience rather than the evolution of a designed "artifact".

A few design environments do accommodate branching, notably the *Designer's Outpost* [45]. They may not exhaustively preserve every state, but they preserve at least the branch points, which is enough to track the development of a design.

Version Control Systems. A related class of software comprises version control systems, such as CVS[36] and its core, RCS [69]. These are, in effect, "disembodied" history mechanisms which preserve and connect versions of files representing states of digital artifacts, especially programs. Such systems (appropriately) do not save states exhaustively; manual check-in is required. However, they offer a valuable feature which Dynapad's history mechanism lacks: the ability to *merge* divergent states. Such merging essentially turns a heredity tree into a multi-tree, a structure beyond the scope of this model.

[36] "Concurrent Versioning System"

The Ambiguity of "Forward" and "Back"

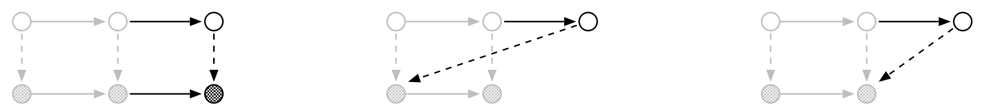
Strictly linear (non-branching) histories, which include most of the applications above, employ a metaphor in which the notions of "forward" and "back" seem apt and intuitive. But in a system which dissociates chronology and heredity, these terms are ambiguous until we specify the connection type. For example, in the scenario of Figure 3.56, suppose we're just returned to state **b** (in view **b'**). Where would "back" go? The previous (most recent) *view* is **c**, but the previous (parent) *state* is **a**.

Even a simple browser creates a similar confusion: once we go "back", we may forget that we must go "forward" to return to the *previous* view. This example suggests that a browser's history is more analogous to heredity than chronology.

Although all ambiguity can be eliminated by having two separate "back" operations, it seems likely that users would confuse them. Therefore let's assume only a single back and forward operation which, following the conventions of a browser, traverses heredity links. These operations can be aptly labeled "undo" and "redo", as we'll see in a moment.

Changes vs. Visits

Nearly all actions a user takes in a history-enabled application fall into one of two categories: *changes* and *visits*.



(a) *Changing* to a new state (b) *Visiting* an old state (e.g. load, undo). (c) Re-*visiting* the current state (e.g. zooming).

Figure 3.59: *Change* and *visit* actions

The majority of normal actions, including moving items and manipulating region-tools, are *changes* which put the workspace into a new state. So each *change* action adds a new state and a corresponding first view.

In contrast, a few operations are *visits*, which add only a new view and restore an existing state. The primary visiting actions are *undo*, *redo*, and *load*.

Undo, Redo, and Load. We might think of "undoing" or "redoing" as changing (or "unchanging" and "rechanging") the state of the workspace. But in either direction, the result is not a unique state but one which already existed. So these operations are more appropriately considered changes in *view*: they extend the *chronological* record but not the *heredity*.

This convention also respects an intuitive parallel between *undoing* a series of changes and *loading* the state that precedes them. In Figure 3.56, for example, restoring **b** can be considered either loading it or undoing **c**. Either interpretation behaves the same, adding a new view **b'** but not a state.

When a reloaded state is an "ancestor" of the previous state, as in Figure 3.56, loading is essentially a long-range *undo*. But loading can also restore "cousin" states on parallel branches; for example, state **c** could be loaded directly from state **d**. Or, if already visiting the past, loading can restore a future state; for example, loading state **c** from view **b'**. Therefore we should consider *undo* as a special case of loading: loading restores any state, and undoing loads specifically the parent state.

A *redo* does the opposite of *undo*, advancing the view to the next state in the heredity. It then makes sense to *redo* without an explicit *undo* but instead after a *load*: just visit the next state after that.

The "Active Future". But of course, because the heredity branches, any state may have more than one "next" state. Which way should *redo* go at a fork? Obviously, if it was preceded by an *undo*, it should return along the same branch. The chronologically previous view dictates an "active future" somewhere ahead of the "present" state. And this principle can be generalized to work with all cases of loading: the *active future* is the farthest visited state such that another branch has not been visited more recently. Redoing always heads toward the active future, or pushes it forward to the next fork.

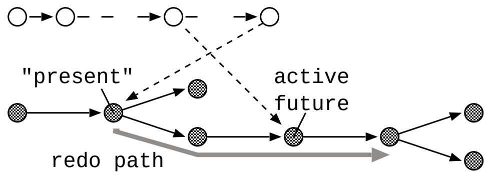


Figure 3.60: *Redo* follows a path to the *active future.*

When the active future is at a fork, and the "present" has reached that state, there is no reliable interpretation of *redo*. Advancing farther requires the user to choose among the possible futures.

Saving. The traditional counterpart of loading is *saving*. But what would that mean in a system which already "saves" every possible state? The utility of normal, manual saving is not merely to preserve a state but also to *name* it, marking it identifiably for later loading, and to make it available to the file system, potentially for sharing with other applications. In the abstract, an interactive historical system may retain a manual *save* operation for either or both of these purposes.

In Dynapad's current implementation, saving a state assigns it a user-supplied name and creates an alias in the file system which is merely a pointer within a *log* (see 3.7.2). Saving a state also converts it into a keyframe (see 3.7.2).

Zooming and Panning. Among the most frequent actions is navigating the workspace by zooming and panning. Should we consider these changes or visits?

Clearly they do not change the workspace; its organization at any state does not depend on any prior zooms or pans. In that sense, navigating contributes to the user's experience but not the artifact. So we should consider such navigation as visits: zooming and panning produce new views (in both the "camera view" and "historical moment" senses) of the current workspace state.

This suggests that every historical visit should include parameters representing the camera view: x, y, and z, as well as the timestamp of chronology.

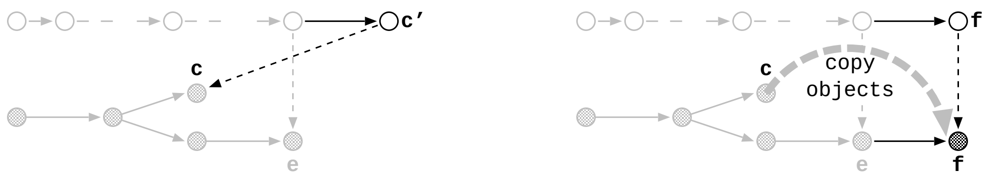
One downside of treating navigation as a visit is that erroneous camera views cannot be corrected simply with an "undo". Here is an example of the possible utility of the other interpretation of "back": *return to what I just saw, without undoing the last action.*

But whether considered changes or visits, navigation actions are added to the chronological record of the user's experience, making them available to an analyst studying that interaction, as Chapter 4 will demonstrate.

Selecting. Finally, should we consider selection a change or a visit? It seems a matter of taste. On one hand, like navigation, selection changes no objects, so we might record it only as a visit. On the other hand, there is stronger incentive to make selection undoable: complex selections are cleared with only a single background tap and may be mistakenly lost and hard to recover. Dynapad's current choice is to consider selection a state-change, which makes it easily undoable.

Restoring vs. Importing

Let's think more deeply about loading. In a loose sense, loading means "getting back" some past state of an artifact. But we can interpret a state in two ways: as a "moment" embedded in a historical context, or in isolation, stripped of that context. Consequently, there are two ways of loading: *restoring* and *importing* (Figure 3.61).



(a) Restoring (visiting) **c** (b) Importing **c** to **f**

Figure 3.61: Two interpretations of loading

Restoring is the sense of loading used earlier: a *visit* to an existing state. Restoring a state leaves it in its historical context and "moves" the user to it. The user's prior context is put aside, and the new context is activated. Undo and redo, for example, then apply to the new context.

Importing a state instead copies it to the user's current context, leaving behind its own former context. That is, all objects in the imported state are *added* to the current workspace. Importing is therefore a *change* rather than a visit; it adds a new state with the combined objects to the local history branch. All prior history of the imported set is stripped away, and they become freshly-minted objects in their new world. Of course, they are copies: the originals still remain undisturbed in their own world and its context.

Merging Branches. In this model, it is impossible for different branches, once diverged, ever to meet again as a single state which inherits the history of both. That is, no state may have more than one previous state — the topology is a strict tree, not a multi-tree. However, any state may imported elsewhere in the same tree (as in Figure 3.61(b)). So divergent branches may be effectively merged by importing one into the other. The imported copies lose their history (although they still remain on the other branch) and become part of the local history.

Each imported object may have a "twin" in its new world, a duplicate of itself that diverged at an earlier fork in heredity. These duplicates can then be linked by brushing (3.4.2) as if they were copied manually.

Copying as Importing. In fact, copying can be considered a special case of importing. Objects copied to the clipboard are stripped of their historical context, then imported (as a "partial" state) when pasted into a new context.

3.7.2 Dynapad's Implementation

Dynapad's current implementation of interactive history, described next, is but one solution to the abstract specification described above.

Many details in this section may be beyond the interest of some readers. The most important idea to take away is that Dynapad encodes the users' activity (both state- and view- changes) in a set of log files which describe the changes that occur between successive states. These logs are processed as part of the ethnographic evaluation of the system (see 4.1.4).

State IDs & timestamps. The state- and view-graphs, like all graphs, are represented by an enumeration of nodes which refer to each other. This requires that each node have a unique identifier: a state (or view) ID. Each new state is labeled with the timestamp of when it was first created, the time of the action which derived it from the previous state. Likewise, each view is labeled with the time of its visit to its state, and also refers to the ID of that state.

Using timestamps not only guarantees unique identifiers, it also encodes the time course of activity, for analysis purposes.

Keyframes. Only certain states, the *keyframes*, can be reconstructed from scratch. Keyframe states encode the details of the entire workspace at that moment. Keyframes represent only starting states, not any changes between states.

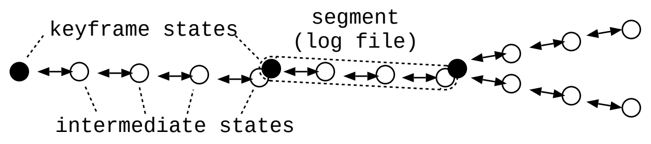


Figure 3.62: Log segments with keyframes and incremental states

All other (*intermediate*, or *incremental*) states (I) encode only the differences from their previous states (P). The difference has two parts: the redo and undo actions. The redo actions are applied when moving forward from P to I, and the undo actions are applied in reverse, from I back to P. This lets the sequence of states be reconstructed moving either forward or backward in time. These redo-undo descriptions represent the edges (two-way links) of the state-graph; each edge is encoded with the node (state) following it.

Branching occurs only at keyframe states (although not all keyframes branch). Non-branching keyframes may occur when states need to be accessed directly, for example, when a state is explicitly marked as a save point.

Segments. A *segment* is a series of states between, and including, successive keyframes. Because branches occur only at keyframes, segments are always simple linear sequences. A segment's first frame is the keyframe which serves as a starting state from which the intermediate states are derived. The last state in a segment corresponds to the keyframe of the next segment but is encoded incrementally, representing the transition between adjacent segments.

Log files. Each segment is written to a separate log file, whose entries encode the states and edges in that segment. The basic structure of a log file is:

(start-state *id build-action*)
(change-state *id redo-action undo-action*)
...

Undo/Redo stacks. Only the current segment is loaded into Dynapad at any moment. The states in that segment are put into two stacks: the Redo (future) stack holds those states after the current state, and the Undo (past) stack holds the current and all prior states. Performing a Redo operation constructs the next state by running the redo-action of the top Redo state. Undo restores the last state by running the undo-action of the top Undo state.

When a keyframe state is restored from a non-adjacent state, the old Undo/Redo stacks are flushed and replaced with the entries in that log segment, and the *build-action* is run to construct the initial keyframe state.

When advancing forward from one segment to another, the keyframe's *build-action* is skipped and the last segment's *redo-action* is used instead, which will arrive at the same state without rebuilding it anew.

Note that each redo-action corresponds to some action in the workspace initiated by the user. By extracting these entries from the log files, an analyst can reconstruct a complete record of the user's activity. The analyses is Chapter 4 are derived in part from such a record.

Branching. Performing any change action adds a new state to one of the segments. If the current state viewed is the last state in a branch (the end of a terminal segment), the new state is simply appended to the end of that segment (Figure 3.63).

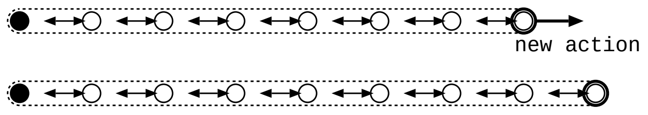


Figure 3.63: Appending to a terminal state

However, changes to non-terminal states will require branching (Figure 3.64). This means that the user is visiting a past state and manipulates the workspace in a way different[37] from the earlier action (Redo) in that state. When branching, the current segment breaks (at the branch point) into two segments; the log file is simply divided in two, and a starting keyframe is generated for the now-independent second

[37]Currently Dynapad does not check whether an action at a non-terminal state is equivalent to an existing transition. Therefore it will sometimes branch unnecessarily, producing identical but divergent states.

part. That same starting keyframe is used to start a third segment, which represents the new, divergent branch. The new state is added as the first incremental state after the keyframe in this new segment, and additional actions follow there.

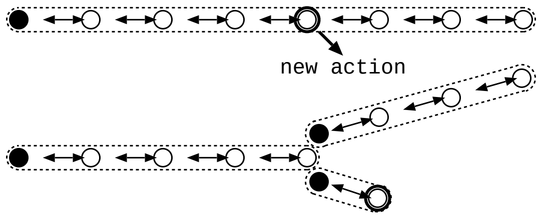


Figure 3.64: Branching at an intermediate state

Saving. Explicitly marking a state as a save point requires that it also have a keyframe, so that it can be restored directly. In this case, the current segment is simply split in two and a keyframe is generated for the marked state at the start of the second part.

View changes. The states encoded in the log files represent the state-graph of the history. The view-graph is represented implicitly in the same files. New views resulting from new states are implied in that state's entry. Visits to old states simply append a special *(visit-state...)* entry to the end of the current segment:

(visit-state *state-id view-id*)

The state-id refers to an earlier state (possibly in some other log) and the view-id encodes the time of the visit (e.g. undo, redo, restore...). While traversing the state-graph by Undoing and Redoing use only the *(change-state...)* entries, the analysis mode follows the *(visit-state...)* entries as well, and may therefore follow a very different course through the state graph.

3.7.3 Visualization & Interaction

Finally, now that we've established some technical constraints on the mechanism, let's examine Dynapad's interface to a workspace's history.

The current interface, described here, is a relatively early design and has a number of problems which can be foreseen even with few observations of real use. Nevertheless, exposure of such problems is an essential component of the design cycle.

Visualization. The visualization of a workspace's history mimics its branching heredity structure with an interactive *history tree.* This tree grows from left to right in the lower left corner of the Dynapad view (Figure 3.65). The tree "sticks" to the view, keeping the same size and position during zooming or panning, and always floats above any objects on the workspace surface.

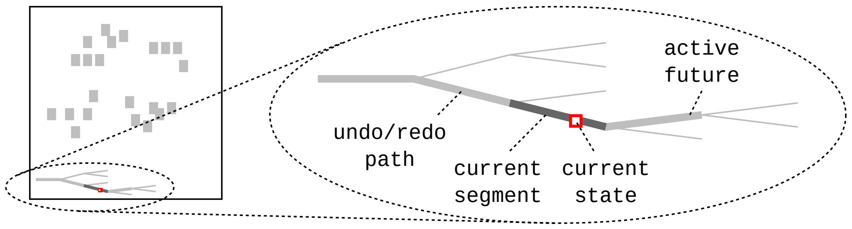


Figure 3.65: Dynapad's "history tree" interface

Each segment of the tree corresponds to a log file, an unbranching sequence of states. Only the first state in each segment (the keyframe) can be loaded directly. A red "bead" marks the location of the current state (which may not be the newest, as in Figure 3.65), and the segment containing that state is highlighted.

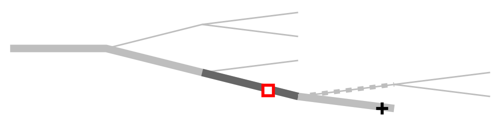
The tree's *active future* state (see above) always lies somewhere to the right of (or directly on) the current-state marker. A path of segments from the tree's root to the active future, which always includes the current segment, appears thicker than other segments and represents the *active path.* The undo and redo operations will move the current state (and the marker) back and forth along this path. Its position within a segment is interpolated according to the number of states in that segment.

The tree may also be hidden and deactivated with a menu option.

Interaction. Beside offering a visual depiction of the historical context, the tree also allows two manipulations:

- A single mouse click on a future segment (a descendant of the current segment) selects a new active future and reroutes the active path to that segment. This is how the user specifies where *redo* should branch if the active path stops at a fork. Figure 3.66(a) shows the result after clicking on the tree in Figure 3.65.
- A double mouse click on *any* segment will *restore* (i.e. visit) that segment's first

state, its keyframe. The workspace changes contents to reflect the visited state and the tree display is updated (as in Figure 3.66(b)).


(a) Single-click on future branch redirects active future

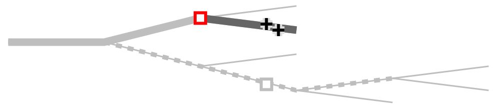

(b) Double-click on any branch visits its initial state

Figure 3.66: Interactions with history tree

Segment Lengths. In the current display, all tree segments are the same length, regardless of the history they span. Segment length is a potentially informative variable, but utilizing it has two hazards. First, segment length has two equally intuitive interpretations: it could represent either the number of states in the segment, or the total time interval it spans. Ideally, the depiction could reflect both duration and density, perhaps by showing a small "bead" for each state, spaced according to their relative timing. But this too is problematic, since the delay between successive states varies by orders of magnitude — sometimes seconds, sometimes months. Linear scaling would be impractical.

Second, as a practical matter, different-length segments makes much more difficult the problem of laying out the entire tree. The layout algorithm should hopefully consider not only the quality of a particular arrangement, but also the *continuity* between arrangements as the tree grow and branches. Animation between arrangements could be used to improve the apparent continuity.

Depicting Timing. States vary in their ages, monotonically along each path from the root to a terminal. Some branches are newer than others, and the newness of a branch varies along its length. This information could be a helpful cue for navigating the tree.

Some information about age is reflected in the vertical position of branches: since new branches "sprout" on the bottom of a fork, ages tend to decrease from top to bottom. But this is unreliable, since it considers only the age of a branch's base, not its tip. Ideally, the relative timing along each branch could be made visible.

An alternative to scaling segments or beads would be to use color or brightness to represent the relative chronology of states. Brightness is more quantitatively expressive [10], but an ordinal representation may be sufficient and could be expressed by limited range of hues — for example, *red=warm=recent*, *blue=cool=old*.

Some Known Problems

Besides the unfortunate invisibility of timing, this interface has several additional hazards:

- The most likely state of a workspace to continue work on is the most recent state of a path, the terminal end of its last branch. But the user cannot jump directly to a terminal state, because a keyframe isn't made until a segment branches into successors. A terminal state must be reached by restoring the last segment's start state and advancing with *redo* to the end.
- All *change* actions (including moving items) keep the current-state marker at the end of its segment. Since segments are the same length no matter how many states they contain, the marker doesn't move and the tree appears unreactive.
- The tree may be mistakenly clicked (or worse, double-clicked) when trying to grip objects near it.
- If the tree segments are too large or widely spaced, the tree intrudes in Dynapad's view and may even run past the edge. But if segments are too close, the structure is hard to discern and clicks may target the wrong branch or miss entirely.
- If the current state is not a terminal state, any *change* action creates a new branch. One chronic cause is missing the tree when trying to redirect the active path, instead clicking the background: if objects were selected, they are then de-selected, causing a workspace change and a new branch.
- Because there is no *chronological* "back" operation (which would return to the immediately prior view), an error in navigating the tree has no easy recovery.

Although the underlying history mechanism has proven valuable, the current interface to that mechanism clearly leaves opportunities for refinement.

3.7.4 The Organizational Impact of Interactive History

To conclude this section, let's summarize how interactive history can impact the cognitive ecology of collection management.

We considered earlier how lenses allow incompatible arrangements to coexist. They do so because of their *proxy* attribute; they duplicate the members of one arrangement to use in another.

Of course, lenses aren't the only way to do this; sets of objects could be copied manually and rearranged in a free space. But a lens reduces the cost of copying and cleaning up temporary duplicates. It also eliminates the cost of allocating new space for the new arrangement, instead superimposing it over the old one.

But lenses also eliminate another hidden cost: the consequence of mistakes. When duplicating and rearranging manually, the user might mistakenly move or even delete originals. Lenses prevent such errors; they facilitate rearrangement not just by saving effort, but by insuring against damage.

As a consequence, lenses offer us greater freedom to explore, to embrace the reflective loop. Instead of trying to imagine the consequences of an arrangement, we can try it out, then react from there, as far as necessary.

Interactive history offers these same benefits. It acts as a giant, omnipresent lens over the entire workspace. It too works by duplicating: each new state effectively copies the objects it inherits, while keeping the originals safe.

Furthermore, interactive history eliminates the effort required to manage a lens: initiating, placing, and sizing it, and perhaps adjusting objects to accommodate its rectangular footprint. Of course, interactive history does induce costs of its own, especially in keeping track of a proliferation of states.

Let's put this explicitly in the terms of Chapter 2. Remember why deferred categorization helps flexibility: it slows the filer from making investments which he is reluctant to undo. But with alternate worlds, users never have to undo any investment. Therefore interactive history *synthesizes flexibility*: it permits multiple organization systems, each of which is equally stable,and supports incremental transition between them.

4

The Structure of Activity in Dynapad

Chapter 1 introduced the core principle of this work, that the activity which arises in any designed environment is a product of both an ecology of the surrounding practices and the cost structure of that environment.

Chapter 2 described the basic cognitive ecology of the practice of collection management, largely abstracted away from any particular environment.

Chapter 3 described a specific environment, Dynapad, and characterized its *functional* structure: the various elements of its design, their rationale, and possible cognitive implications.

Finally, this chapter will examine the convergence of these two influences: how the ecology of collection management and the cost-structure of Dynapad shape the participants' behavior. The emergent structure of activity described here is influenced by the structure of the design, but they are not homologous. That is, the meaningful units and patterns of behavior do not correspond to the elements of the design. We should not expect, for example, independent strategies for "brushing", "lensing", "traying". Rather, such functional elements are combined in unpredictable ways as users negotiate their collections. However, the strategies they adopt are shaped by the *framing* of the design, the actions that suggest themselves from the visible functionality — that is, Dynapad's cue structure.

4.1 Methodology

The user study described here is in the style of Malone's *exploratory observation* [49]. It is intended neither to measure Dynapad's utility nor to document exhaustively the huge space of possible user activity. Instead, it draws on a small but high-fidelity sample of behavior in order to elicit important phenomena and trace the impact of design details.

4.1.1 Participants

I observed two subjects, **A** and **B**, over a combined 12 hours of interaction with Dynapad. Subject **A** was a male senior undergraduate, a Cognitive Science major specializing in HCI. While he had no particular expertise in the research topics specific to Dynapad, he had social and intellectual investment in our shared community. Subject **B** was a female graduate student in Cognitive Science. Her own primary research at that time was peripheral to HCI, but she had had some exposure to and interest in HCI-related topics.

Neither subject had any prior experience working with Dynapad, although both had seen brief demos of various implementations. Both seemingly had the same motivations for participating: first, a collegial generosity to me as a research colleague; second, a curiosity about Dynapad as a novel and engaging application; and third, a self-interest in organizing their own document collections.

This third motivation was amplified by the fact that both subjects were planning soon to leave the department (**A** by graduating, **B** for temporary employment elsewhere). So for both, their personal information workflow and their relationship to their collected material were likely to change. They recognized this study as an opportunity before such changes to solidify their memory of their collections and to harvest value from them.

The Collections

Currently, Dynapad handles only images and PDF documents, which are displayed as portraits (see 3.1). Since our earlier work [3] explored photo collections (see 3.5), this study included only PDFs.

A central theme of this research is to support *personal* digital collections. Although some research has employed digital content unfamiliar to the participants (e.g. [31]), we believe it is crucial that users care about and engage their own materials, as **A** and **B** did.

Subject **A** contributed essentially his entire personal file system, first converting all MS Word and PowerPoint documents to PDFs using a third-party batch converter. **A** himself had composed most of these documents, which included coursework, class presentations, business letters, resumés, and even temporary "scratch" documents. Dynapad imported both these PDFs and **A**'s native PDFs, which included research papers, course handouts, magazine articles, and commercial forms. Overall, **A**'s collection comprised 353 PDFs spanning a wide variety of content, including considerable "junk".

Also, the collection inherited the structure of **A**'s file system, and **A** would later use that initial structure guide his Dynapad organization, as we'll soon see.

In contrast, **B**'s collection was smaller (212 documents) and more specialized. **B** included only documents already in PDF form, which were mostly academic publications related to her research or graduate courses. Unlike **A**, **B** simply retrieved all PDFs in her file system, and the resulting batch, lumped together, lost any associations it might have inherited from her directory structures.

B imported her collection into Dynapad in two batches: 155 initially, then 57 more which she had acquired since the first session.

4.1.2 Participants' Goals and Expectations

I intentionally did not give these subjects a well-defined task in Dynapad. Instead, the goal of their activity was negotiated gradually in our conversations preceding and during their sessions. The gist of my instruction was essentially this:

> Explore your collection. Organize it to the extent that you find useful. Consider this an opportunity to harvest some sort of value from what you've accumulated.

Their lack of explicit initial goals meant greater *reactivity*, response to opportunities which emerged in their workspaces. In terms of the framework presented in Chapter 1, we can think of this as weighting more highly their reflective "sensemaking" loop (Figure 1.10), relative to any fixed goals established by their loose contextual "practice" of "harvesting value".

Over time, **A** and **B** did establish more precise goals for themselves. For example, **A** eventually decided to use his organization to build a portfolio of his best work to showcase for potential employers. We might regard this as an example of negotiated usage (Figure 1.13).

Increased Metacognitive Demands. But this lack of guidance also left the subjects more responsible for reflecting on their own processes: inventing strategies and monitoring their effectiveness, and recognizing when their own goals changed or were temporarily interrupted.

4.1.3 Observing and Recording Sessions

The protocol for these sessions was similar to that of the earlier PhotoPad sessions (see 3.5), in that each subject worked collaboratively with an interviewer. In PhotoPad's "wizard-of-Oz" protocol, the interviewer was also the operator, controlling Dynapad at the subject's instruction. In this case, however, the subjects themselves were in control of Dynapad, while the interviewer (**I**, the author) was at hand to assist, answer questions, and maintain a dialog to elicit continuously the subject's thoughts.

As before, Dynapad was projected onto a tabletop over which the participants could gesture throughout their conversation. However, **A** and **B** used different input devices. **A** used a normal mouse on the table. **B** used the DiamondTouch table [17] (see 3.3.4), where the projected workspace was superimposed on the input surface, giving it a feeling of directness.

Data Collection. With both subjects, audio and video was recorded with a overhead camera which captured both the projected display of Dynapad and the participants' gestures on the table.

In addition, Dynapad's history mechanism (see 3.7) kept logs of all activity. This richly detailed data source supplemented the video recordings and facilitated analysis in two ways. First, since the history was exhaustively functional, any state the subject experienced could be revisited and exported to produce the detailed figures in section 4.2 ahead. Second, the history logs can be converted to readable transcripts of each session, which can then be further annotated from the video record. Details of this transcription process are discussed below (4.1.4).

Training

Since both subjects were new to Dynapad and needed to operate it themselves, they received some training. Before working with their own collections, they each spent about 50 minutes (session A0/B0) with a sample PDF collection while learning Dynapad's features and interface, under the interviewer's guidance. For subject **B** especially, using the newly-prototyped DiamondTouch interface, this was also an opportunity to uncover some bugs which were corrected before the next session.

Session Chronology

Table 4.1 summarizes the duration and separation of each subject's sessions. Note that all of **B**'s sessions precede **A**'s, although I will begin with **A** in the discussion ahead.

Table 4.1: Chronology of subjects' recorded Dynapad sessions

Session	Day #	Duration	Notes
Subject **B** (DiamondTouch table interface):			
B0	0	0:50	training and debugging only
B1	3	0:52	imported own collection of 155
B2	11	0:54	
B3	24	0:55	imported 57 more docs in mid-session (0:33)
B4	25	0:56	
	Total:	4:27	
Subject **A** (mouse interface):			
A0	0	0:49	training only
A1	0	1:14	imported own collection of 353
A2	1	1:11	
A3	2	1:01	
A4	16	1:23	
A5	117	1:39	added labels to piles
	Total:	7:17	

4.1.4 Preparing Transcripts

Logged Actions

At the time of this study, Dynapad's log files recorded the following types of actions:

1. Creating, copying, or deleting any objects (e.g. PDF portraits, region-tools);
2. Moving any objects;[1]
3. Selecting and de-selecting objects;
4. Resizing region-tools;
5. Panning and interpolated zooming (i.e. by pushing or pulling). If the zoom target is an object, it is identified.

The following events were not logged:

1. "Instant" zooming by double-clicking;
2. Open or closing the document-viewing application, and any activity within that application (e.g. browsing pages);
3. Unlocking, rearranging, and re-locking document portraits;[2]
4. Rolling over (brushing) items which highlight others;
5. Popping up menus, and invoking any menu command not otherwise logged;
6. Initiating selection boxes or lassos which don't change the existing selection.

Most or all of these unlogged events *should* be logged as well; the implementation was merely incomplete.

[1]Dynapad recorded moved objects' initial and final positions, but not their paths or timing. Such details are not usually important but sometimes reveal meaningful behaviors such as *hovering*: pausing while holding the object over a spot that is subsequently rejected.

[2]Although not explicitly logged, unlocking and rearrangement could be inferred from other logged events (e.g. selection and movement of component images).

Synchronization

The log transcripts are easiest to use as a supplement to video recordings if they are indexed to the time code on the video. To synchronize them, each video segment must be aligned with the logged history at a single event that can be identified clearly on both the video and in one of logs it spans. The timestamp t of the event's *(change-state...)* entry is combined with the time v on the video in a *(tape-synch s v)* entry, which is then inserted into the log at the position corresponding to the video start, as Figure 4.1 depicts.

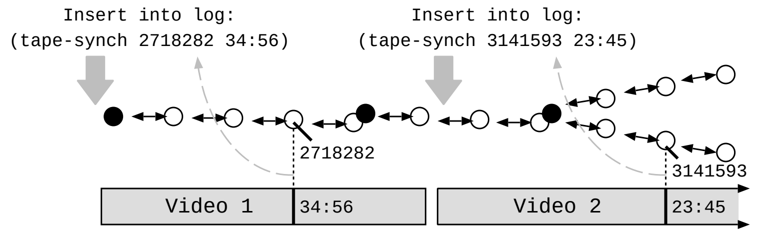


Figure 4.1: Synchronizing video recordings with log files

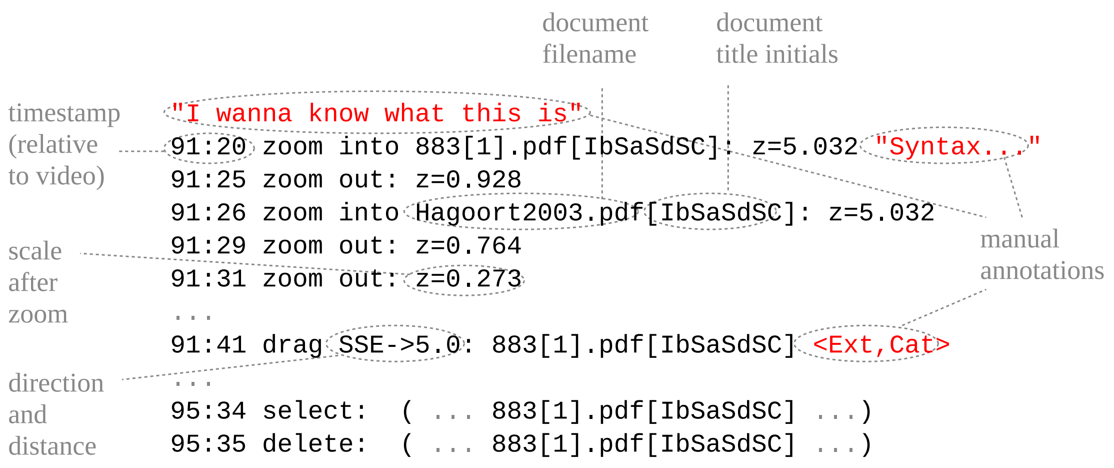
Summarization

The raw log files are machine-readable, executable Scheme code. A special Dynapad utility[3] converts them to a more human-readable format. This format omits some details and converts others into a form that makes each event easier to identify in the video recording. For example, movement and panning vectors are expressed in portrait-lengths and 16-point compass directions, and documents are identified with an abbreviation of their title, which can often be seen in the video. Figure 4.2 shows a short excerpt of such a transcript and describes the scenario it encodes.

Once these initial transcripts are generated automatically, they provide an accurate and detailed framework for further annotation by hand of information from the corresponding video.

Furthermore, an analyst can replay sequences of events within Dynapad itself by stepping through portions of the exhaustively saved history, actually running the code in the log from which a transcript is derived.

[3]log-summary.ss



```
"I wanna know what this is"
91:20 zoom into 883[1].pdf[IbSaSdSC]: z=5.032 "Syntax..."
91:25 zoom out: z=0.928
91:26 zoom into Hagoort2003.pdf[IbSaSdSC]: z=5.032
91:29 zoom out: z=0.764
91:31 zoom out: z=0.273
...
91:41 drag SSE->5.0: 883[1].pdf[IbSaSdSC] <Ext,Cat>
...
95:34 select:  ( ... 883[1].pdf[IbSaSdSC] ...)
95:35 delete:  ( ... 883[1].pdf[IbSaSdSC] ...)
```

Figure 4.2: Sample log transcript excerpt.
The scenario for this excerpt is as follows: initially, **B** has several documents in a timeline tray, sorted by publication date. Some are twins, the same paper appearing twice with different file names. One such pair is a document entitled "Interplay between Syntax and Semantics during Sentence Comprehension" (abbreviated to "IbSaSdSC" in the transcript) with file names "883[1].pdf" and "Hagoort2003.pdf". There is nothing linking them together, but because they have the exact same publication date, they are adjacent in the tray and easy to spot. **B** zooms into each, confirming that they're identical, moves one downward with some other duplicates, and later selects and deletes those duplicates.
The manual annotations include **B**'s speech and tags (**[Ext,Cat]**) identifying the function of that move. For a full explanation of these tags, see 4.2.

In constructing the narrative of my subjects' activity in the next section, I have relied primarily on the video record, but used partial transcripts and replayed certain scenes to verify details.

4.1.5 External Validity and Ethnographic Practice

As a final note, let's situate this methodology with regard to other ethnographic practices.

True ethnography takes the analyst completely out of the laboratory and into a natural environment, observing contextualized and self-motivated behavior which he tries not to influence in any way. Clearly the observations reported here do not meet that standard, for three reasons.

First, this study was indeed conducted in the laboratory. And although we might argue that the relevant arena of behavior is not the physical setting but the virtual "inner environment" of Dynapad, that too is a *designed* environment rather than a natural one.

However, this study is naturalistic with respect to its *holism*; it makes no attempt to isolate individual variables or to control or eliminate sources of complexity. On the contrary, it aims to document that complexity, to consider the impact of a design as an interconnected system of cues and affordances.

Second, the subjects' activity did not arise on its own but was effected for the purpose of this research. It's unlikely that these subjects would have invested effort to organize their document collections without the opportunity presented by this study, and certainly they would not have done so in the same way. In that sense, this research is studying activity that it has uniquely *manufactured*.

And yet, as Chapters 1 and 2 have suggested, this activity inherits momentum from established practices. The subjects had already collected their documents for other purposes and had a degree of self-motivation from their personal relationship to the content. Dynapad did not create a need for organization, but merely provided a novel means to address it.

Third, as a researcher, I was not strictly a passive observer but a participant. However, since I gave few specific instructions and generally avoided intervening, the vast majority of the subjects' actions were self-initiated, and many demonstrated strategies or misunderstandings that surprised me as both interviewer and designer. Those are the discoveries this study means to harvest.

4.1.6 The Analysis Ahead

The remainder of this chapter documents my observations of **A** and **B**. The narrative is roughly chronological and is concerned primarily with the emergence and causes of large-scale phenomena, particularly the subjects' gradual organization of their workspaces. It also offers a context for the various examples of micro-behaviors.

4.2 The Development of Organization

This section offers a detailed narrative of two subjects, **A** and **B**, as they used Dynapad to organize their own document collections. The events described are in almost strictly chronological order so as to contextualize each episode with its preceding history. As a result, common themes will recur at several points, and more general discussion of them will be inserted where appropriate.

Transcript/dialog conventions. Much of the narrative ahead will include dialog between the interviewer (**I**, the author) and subject (**A** or **B**). These are included to illuminate the subjects' experience without falsely attributing to them intentions or understanding. Such dialog is annotated with the following conventions:

<*object*>**:** The referent (usually a pile) of a deictic word (e.g. "this");

[**action**]**:** An action concurrent with speech;

(paraphrase): An implied or indistinct word or longer paraphrase;

...: A pause, which may represent omitted dysfluencies (e.g. hesitation, repetition, irrelevant interjections).

All other punctuation is used normally to improve readability and may not reflect the actual dynamics of the utterance.

4.2.1 Subject A

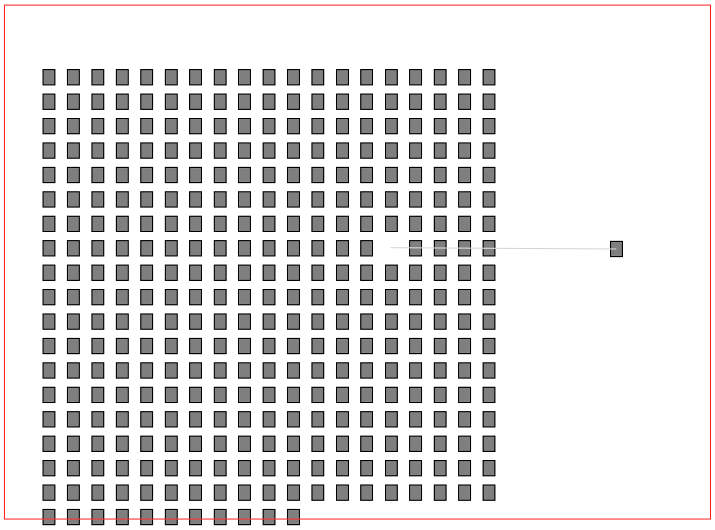
Both subjects' early decisions have far-reaching consequences for the character of the activity following. The initial moves are both easy and difficult: easy since it seems not to matter, but difficult because the slate is blank, with little structure in the artifact to "read" and inform these decision. For this reason, I'll focus especially on **A**'s early activity.

Incoming Organization. Importantly, **A** (unlike **B**) has already invested some structure into his collection on his own file system. With a few exceptions, his practice had been to create a new directory in each academic quarter and put there all new documents (both his own work and others') during that period. In some cases, he had created additional subdirectories for particular topics or courses.

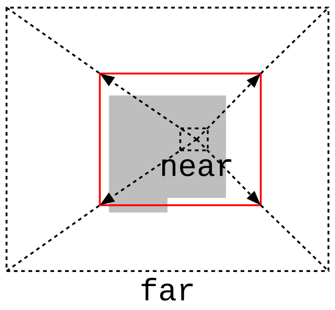
When **A**'s collection is imported into Dynapad, this organization, primarily chronological, is reflected implicitly in the order of items in the initial grid and is remembered as a brushable relation (hidden initially).

Session A1: Inventing Structure

First, **A**'s collection was imported as grid of portraits (as described in 3.4.1), although with directory-brushing turned off (until later in the session, where noted below). Within a few seconds, **A** took his first action: he chose one item, zoomed into it (and also opened a document browser) to identify it, zoomed back out, and extracted it from the grid to an empty space on the right (Figure 4.3(a)). Given the homogeneity of the initial workspace, *some* such action is inevitable. But like any *particular* action, it is nearly arbitrary and yet far-reaching, the first random pebble in an eventual landslide.



(a) Claiming nearby open space



(b) Interpolated zooming finds nearest free space first

Figure 4.3: **A**'s first extraction *(A1:02:28)*

This simple action accomplishes several things at once:

- It establishes a new category at its destination, which begins to determine the overall organization;

- It annotates the item as member of that category;
- It consumes some of the nearby open space – a resource which, although infinite, is not interchangeable;
- It marks the item as processed by removing it from reservoir;
- It leaves a hole, which serves as a landmark in the reservoir;
- It establishes a tentative workflow.

All of these facets have implications for future actions, as we will see. This particular move was determined by two decisions: 1) *why that item?*, and 2) *why that destination?*

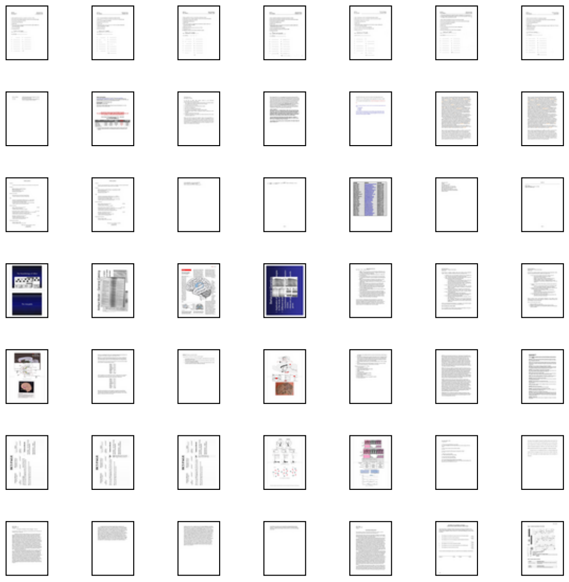


Figure 4.4: The first extracted item (center) among its neighbors.

There is no explicit evidence to answer the first question, but I'll guess: the item (centered in Figure 4.4), a slide with a blue background, was among the most visually salient in a field of mostly white neighbors. And although the details of its portrait might vary, even more volatile is its position in the grid, and therefore the neighbors it competes with for **A**'s attention. Since **A** was only beginning to invent a workflow for himself, and had not yet had time to formulate a strategy beyond simple reaction, Those nearly-random factors may have been significant. Later we'll see how **A**'s selection criteria develops.

Appropriating Free Space. Now for the second question: why did **A** put the item there? One obvious explanation is that it traveled the shortest distance (plus "padding") to empty space. But also, it went to center of the *largest* empty space visible at that moment. The red rectangle in Figure 4.3(a) shows the edge of that view, where **A** stopped zooming out. That view results from interpolated zoom-out (see 3.3.1) from the "near" focus on the object to the "far" re-centering of the grid (Figure 4.3(b)). Because the selected item was to the right of the grid's center, the first open space appeared on the right.

Fortunately, this effect of interpolated zooming will produce the closest possible space for any item in the grid. But it does so only as an *accident* of the layout which breaks down in more complex conditions. For example, as items are placed farther to the right of the grid, zooming out will leave the grid further to the left, biasing space toward the right even when it is not the closest.

What this reveals is that zooming has a second, implicit function besides navigation: to locate and apportion a resource, open space. Dynapad could be designed to dispense this resource less arbitrarily and more judiciously, but it does not. Therefore the assignment of meaningful space is impacted by an *unmitigated side-effect* of zooming, a feature designed for other considerations. The current design is not necessarily inadequate, but it is *indifferent* to an interaction of great importance.

Developing Selection Criteria. Partly as a result of his initial action, **A** next begins to develop more reflective criteria for choosing and placing items. Here is a sample of his actions in this phase, in relative order:

Related Neighbors: With a salient hole in the grid and a category (107c, a course) in mind, **A** focuses near the hole to find other 107c documents and brings them alongside the first.

Extended Category: Aware that the initial grid is partially sorted by academic quarter, **A** sees the potential for re-organizing it by topic:

1 **A:** Hmmm... so instead of organizing things by time, I could just make a "neuro" folder (which would include courses 107a,b,c).

Opportunistic Discovery: **A** spots his own paper (a Contextual Design project) in the lower left, changes focus and starts a new category *HCI* with it. Note that even though this document was found toward the left, the new pile was started to the right, as before. (I don't know the reason, but it wasn't a reaction to whitespace; **A** started dragging the item rightward before ever zooming out.) Once the second pile was established, **A** separated both, moving *HCI* to the lower right corner of the grid and *neuro* to the upper right.

An important question remains: why did **A** classify this new document broadly as *HCI*, rather than more narrowly, perhaps as a course (like 107c)? His explicit articulation of "HCI" as a category may have biased future classifications too broadly, which leads to a big mess later.

In fact, this pile becomes the largest and arguably the messiest of all piles. The story of that pile is the central thread in our narrative of **A**'s activity.

Assimilation: With a couple of exceptions in mid-grid, **A** returns to the growing central hole as a source of documents, but now distributes them primarily to *HCI.* This strategy is a hybrid of recent experience, using the old landmark with the new category. The fact that *HCI* is a broad category makes it easy to find members even in the dissociated location, but this potentially muddies the category further.

Notice that **A**'s initial placements in *HCI* are tidy; this will eventually change as the pile becomes increasingly chaotic.

Third pile: **A** finds a recommendation letter and starts a *misc* category, situated at a third corner, the upper left. The next several documents aren't from courses, so they go to new *misc* rather than differentiating new categories.

A theme emerges: **A**'s dominant categories developed very early, based on a very limited sample of the collection, and quickly became overly broad and disorganized. This seems like an example of *premature filing* (see Figure 2.4), establishing categories with too little experience of their utility and impact.

Serialization: Once his attention is at the upper left, zoomed in, **A** shifted to drawing sequentially from that corner. The next four extractions follow this strategy, leading to the state shown in Figure 4.5.

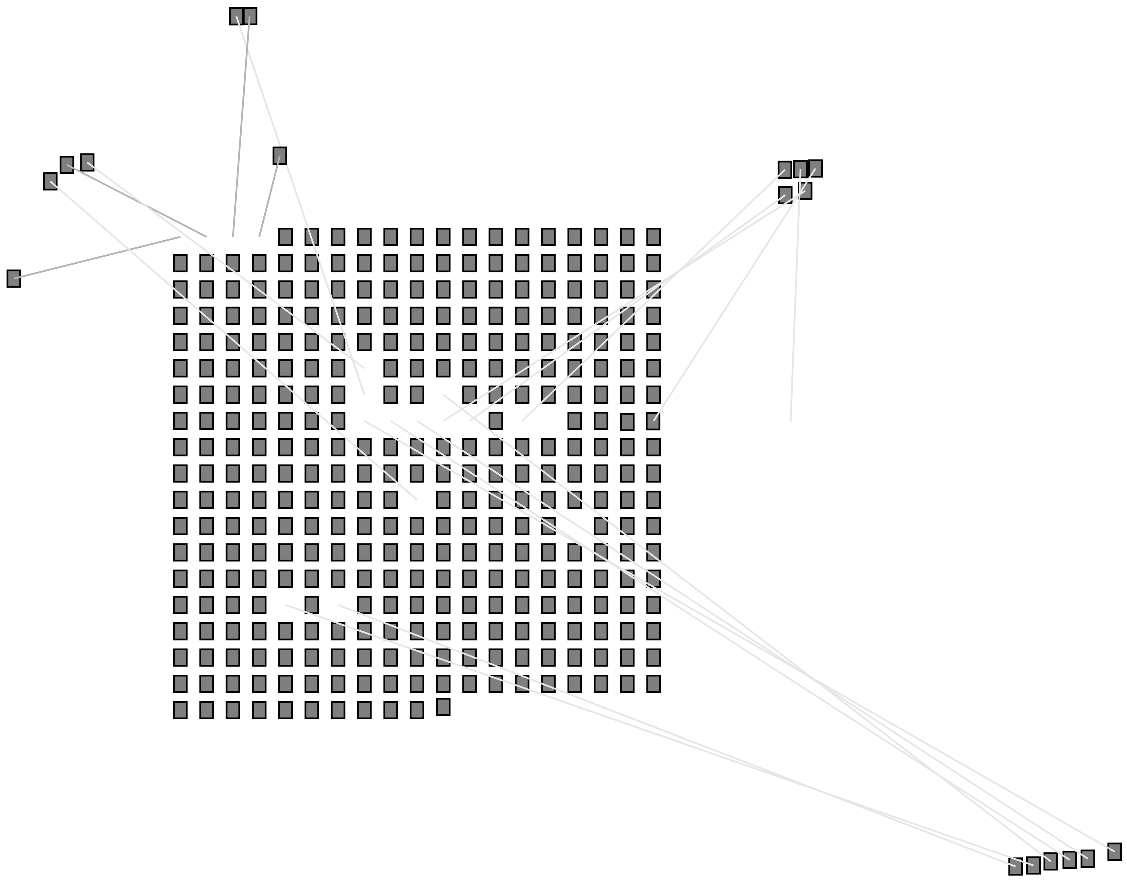


Figure 4.5: The most recent 4 extractions have been in sequence from the upper left

Early Volatility. We should note at this point that, so far, **A**'s workflow has been dominantly *reactive* rather than deliberated. **A** has structured his activity by responding to structure of the environment: details of individual documents, established piles/categories, and the geography and accessibility of the reservoir. **A** is still inventing both his workflow and organization scheme, shifting rapidly between strategies, as the minimal workspace structure changes significantly from small manipulations.

Deliberate Restructuring (scene 1). Suddenly **A** stops his serialized extraction from the upper left, and returns to seeming randomness. **I** asks why:

2 **I:** How are you deciding what to look at next? Are you just... grabbing randomly, or do you have... a system in mind, or what?

3 **A:** I noticed... when I was going linearly [gestures at NW corner] that... things tend to be clumped together... You know when you (import), then... it's probably just going... subdirectory... get everything...

4 **I:** (Explains layout: items from the same directory usually appear together.)

5 **A:** Yeah, so, the reason I'm doing it randomly is to break that up... Because if I go linearly, it'll turn up, [zooms into next item at NW]... yeah, see, look... that's gonna go right there [places it next to last extraction], and the next one [zooms in], if it's the same thing, it would go right there too. I'm pretending that, you know, I just have a lot of data, and it's all cluttered, and... you know... I'm just like, "OK, I'm making piles everywhere."

6 **I:** So you're intentionally hiding the categorizations you have now?

7 **A:** Yeah (laughs)

8 **I:** Because you're afraid you'll just wind up with the same old categories anyway, and you just want to break out of the mold?

9 **A:** Yeah, I don't... want to see things that way.

Beginning to reflect on his own process, **A** has anticipated the value of *organizational flexibility* (see 2.2.1), which is facilitated by the low cost of creating new, implicit categories. Although the initial layout grid was a design choice intended to soften the existing structure, **A** has not only perceived it but actively tries to compensate by randomizing his selections.

Keeping Oriented. Almost always, **A** must zoom into a portrait to identify it (sometimes also opening a reader on the PDF itself). But zooming into items in the largely-intact grid has the danger of *post-ascent disorientation* (see 3.3.2), where **A** loses track of which item he just viewed.

Perhaps as a compensation for this, **A** briefly adopts a new strategy: pulling an item out of the grid even before identifying it. After extracting the bottommost document in Figure 4.6, **A** comments:

10 **A:** I think I'm gonna use this bottom space as my little active work area

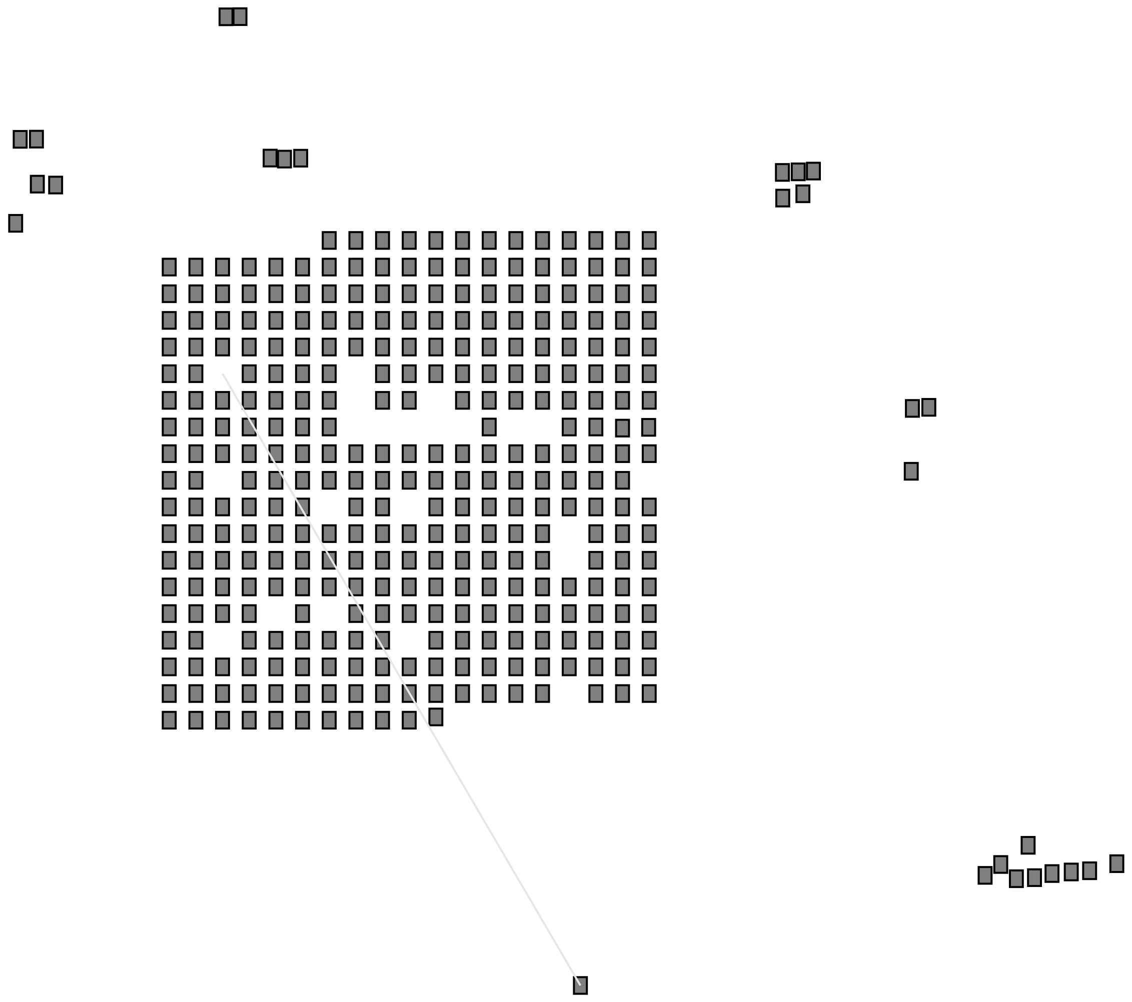


Figure 4.6: Extracting to "active work area", leaving another hole *(A1:26:44)*

He then begins to extract items to below the grid, zooming into them, and then moving them into a category.

Before long, however, **A** became very proficient at carrying while navigating (see 3.3.2): gripping a document while still close, zooming out while holding it, and moving it to its destination. With this ability, and as the grid became more diffuse, **A** mostly returned to examining documents in place.

Deliberate Restructuring (scene 2). Figure 4.6 also shows the context of some additional comments about **A**'s randomization strategy:

11 **A:** [While extracting the last item:] See, I pick a spot that doesn't look... holey.

Several minutes later, discussion continues:

12 **I:** Why are you pretending that (the collection) is random; why not deal with it as it is, as partially structured?

13 **A:** Because this way I'm forced to deal with more piles at a time...

14 **I:** (But you're effectively making it harder on yourself, yes?)

15 **A:** I guess I didn't give it that much thought.

But now **A** seems to be giving it more thought. A minute later, he continues:

16 **A:** Also, if... I do things in multiple spaces [gestures to piles], or... I have more piles, it kind of makes you think about things more... If I just go serially, then it's just gonna be like, "OK, this goes here" [gestures from grid to *neuro*], "OK, this goes here" [repeats gesture], "OK...". It would keep going in the same pile. But... if I just do it more randomly, then I'll look at something and I'll be like, "now I have more options" [quickly taps several piles], right? I have more piles sitting around, and then it's like I have more options to place this one thing. Whereas if I just go serially... later on I would have a huge pile... just directly from this row [gestures to grid], and then I would have to go into (that) pile anyway, and then look at this stuff and be like, "OK, now I want to break this pile up". I guess I'm breaking up the piles from the beginning.

17 **I:** : (To rephrase: you expect to have more or different categories, so instead of first emulating that and then subdividing, you want to start with subdividing?) You want to see it with... fresh eyes?

18 **A:** : Yeah. I already know the old way I did it, so doing it that way is not gonna add new value for me... I'm just trying to think out(side) of the structure I'm already in.

A seems to be making three different points:

1. Randomizing his selection eliminates the temptation of following a rote procedure (i.e. follow the last item) which could become a substitute for re-thinking each item. That is, **A** is deliberately *increasing cognitive effort* at the *source* of an extraction.

2. A greater number of piles makes visible a greater number of distinctions and potential categories. It seems that **A** is deliberately "fertilizing" his workspace with diversity to remind him to diversify, increasing cognitive effort at the *destination*

of an extraction. Or perhaps more precisely: **A** increases the effort of classifying items at all, but decreases the effort of classifying them *thoughtfully*.

3. **A** expects to be able to make more (or at least different) distinctions than those of the initial directory-classes. Making those distinctions from the start will *save mechanical effort* of assembling piles which merely reify the existing groups.

Ironically, it happens that in "seeing with fresh eyes", **A** actually makes *fewer* distinctions than before. Specifically, he makes piles with definitions that are too broad, especially *HCI* and *non-HCI* (what was originally *107c* and then *neuro*). Despite **A**'s explicit intention otherwise, he ends up producing piles that still need considerable subdivision. As we'll consider in detail later, **A** ends up discarding order and must invest more effort to compensate. However, **A**'s efforts do have value: although he conflates many existing distinctions, he does create new ones.

At this point, **I** tries to learn more about how exactly **A** is using the old categorization:

19 **I:** Now... it is true that you're not really using the old way at all; it's as if you hadn't done (it). Is that the case? Or... do you know implicitly that "these are a clump, and I know that I made this a category", and you're quietly assimilating that information? ... For example... [turns on directory brushing] so, maybe you... know that this line of stuff [rolls over row] is all a category that you made before. *Do* you know that, and how are you making use of that information?

20 **A:** I'm trying to ignore it.

Given **A**'s claim to ignore the relation that brushing makes visible, **I** offers to turn it back off, but **A** chooses to leave it on, to "play with it". It remains on through the rest of **A**'s session, and as we'll soon see, he does make use of the signal in unexpected ways.

Deliberate Restructuring (scene 3). Much later in the first session, **A** offers another reason for re-categorizing:

21 **A:** [While moving/classifying a paper on the hippocampus, obtained in course 107A:] I guess the existing file structure I have right now... is probably not as useful... because it's... I only think of things chronologically... So if I wanted to know something about the hippocampus... it's not like I only learned about the

hippocampus in 107A, I could also learn about it in 107C, so I would actually have to jump to two folders.

22 **I:** So you want... hippocampus cross-referencing!

23 **A:** Yeah.

Although **A** has acknowledged the need for *multi-filing* and has been instructed on copying documents, he doesn't do so in this case. The reason may be that he has already decided to use a single pile for 107A, 107C, and other neuroscience courses; filing by subject instead of course would put it in the same place anyway. He could begin making the distinction now, but he continues with the momentum of his established categories.

Dealing with Implicit Organization. Earlier, soon after the state shown Figure 4.6, **A** comments on how he remembers his organization scheme:

24 **A:** After... I have a certain amount of piles, I won't remember... what this <*HCI*> pile is.

25 **I:** Labels would help.

26 **A:** It might need labels, or maybe I could just go like that [zooms to random element of HCI pile, then backs off slightly to see neighbors]... since it is all visual, then I see this image, right? [points to distinctive neighbor of zoom target] Then it's like... "OK, there's my HCI thing", so therefore this is my HCI pile. So... I'm not sure if I would want a label... 'cause then... the overhead of that is that I have to type things. And then, you know, while you're... brainstorming and organizing stuff, then... I would have to continually retype, and I think I just like how I can just... use the mouse.

27 **I:** And just leave piles (as) implicit categories?

28 **A:** Yeah.

A's action here shows zooming being effective at what it was designed for: allowing a low-effort glance to focus and an easy return to context. If visibility is measured not merely as non-occlusion but as the ease of seeing what is needed, then zooming effectively increases visibility.

A's comments address several points about the cost structure of piling:

- The visibility of items in an unstacked pile, amplified by zooming's easy glances for detail, lets salient items remind **A** of the pile's meaning. Visibility improves clarity (Figure 2.11), **A**'s understanding of what spaces mean. This compensates for the ambiguity of implicit organization, a lack of labels.

Naturally, this is most effective for piles which *have* a coherent theme suggested by individual items. As an unfortunate counter-example, later on **A** forgets which is his "trash" pile, and digs through its members looking for a non-existent semantic theme. A "trash" label would have avoided that episode.

- Explicitly labeling the organization incurs a mechanical cost (Figure 2.2) of repeatedly moving to the keyboard and typing. This effort is certainly less than that of labeling physical files, but still **A** prefers to avoid it.
- **A** implies that this mechanical effort would negatively impact "brainstorming and organizing". We can imagine that during such brainstorming **A** would be less likely to rethink a category and change its label if forced to retype it, especially with an existing label reinforcing the old meaning. In that sense, mechanical difficulty hinders flexibility (Figure 2.13).

By **A**'s own assessment, labels aren't necessary at this point. But later the demand will increase with the number of piles, and **A** will respond with an inventive solution.

Use of Brushing (scene 1). Remember that brushing occurs when the cursor overlaps any portrait — which is always true when portraits are gripped and moved. Brushing may be initiated deliberately, but once turned on it happens automatically during all extractions. After one particular extraction (of a paper on Columbus), **A** noticed his reaction to the ambient signal:

29 **A:** Hey, you know what – this highlighting thing is actually useful. Because, you know how I was saying (it would be useful to have labels)?... OK, so this <*Columbus*> was right here <*in the reservoir*>... I saw it, and... I was saying, "Oh yeah, that's my history paper." So I zoomed out, and I highlighted it... And that <*another history document, already extracted*> lights up,... so it's gonna go in that pile.

In this case, **A** chooses to maintain a category (*history*) he made earlier as a directory. But he uses the highlighting not to discover that the category exists, but to *locate* the pile he's already begun. We can also infer a second value of the highlighting: it lets **A** *validate* his identification of the document. For instance, if he decided that it was history but saw his neuroscience pile light up instead, it would afford an opportunity to correct a possible mistake.

Making Piles Explicit. Throughout session 1, **A**'s "piles" remained implicit, close groupings of items without explicit wrappers. One of his last actions of the session was to wrap these groups into explicit region-tool clumps. His motivation was extrinsic: **I** suggested it as a pragmatic response to a bug while saving his workspace. (Wrapping the piles explicitly would have made it easier to recover them if something went wrong with the log files.)

In hindsight, it was unfortunate not to discover when **A** would have made clumps on his own. Starting in session 2, he began to use the affordances of the clumps (portability and insulation), but I cannot tell to what extent he did so because he *had* them or the workspace changed enough that he *needed* them.

Figure 4.7 shows **A**'s workspace at the end of session 1, where he started in session 2.

Session A2: Building Momentum

Local Consolidation. Early in session 2, **A** adopts a new strategy, visible in Figures 4.8–4.10. He has abandoned the randomization of his previous session. Instead, harnessing the grid's partial organization, **A** first gathers together nearby related items into a dense *batch* and only then hauls the entire batch off to a final destination.

Regarding the batch of Figure 4.8:

30 **I:** So what's this category you're building?

31 **A:** HCI [circles local group]. Which is gonna be... [zooms out, points to *HCI* pile] in this group.

32 **I:** So, why are you... doing it there <*locally*>?

33 **A:** So I don't have to come all the way here <*HCI*> and drop it.

A adds later:

34 **A:** See...when you start finding things in that area, it's like, OK, so there's probably more things around here. So I might as well gather around here <*locally*> before traveling all the way there <*destination pile*>.

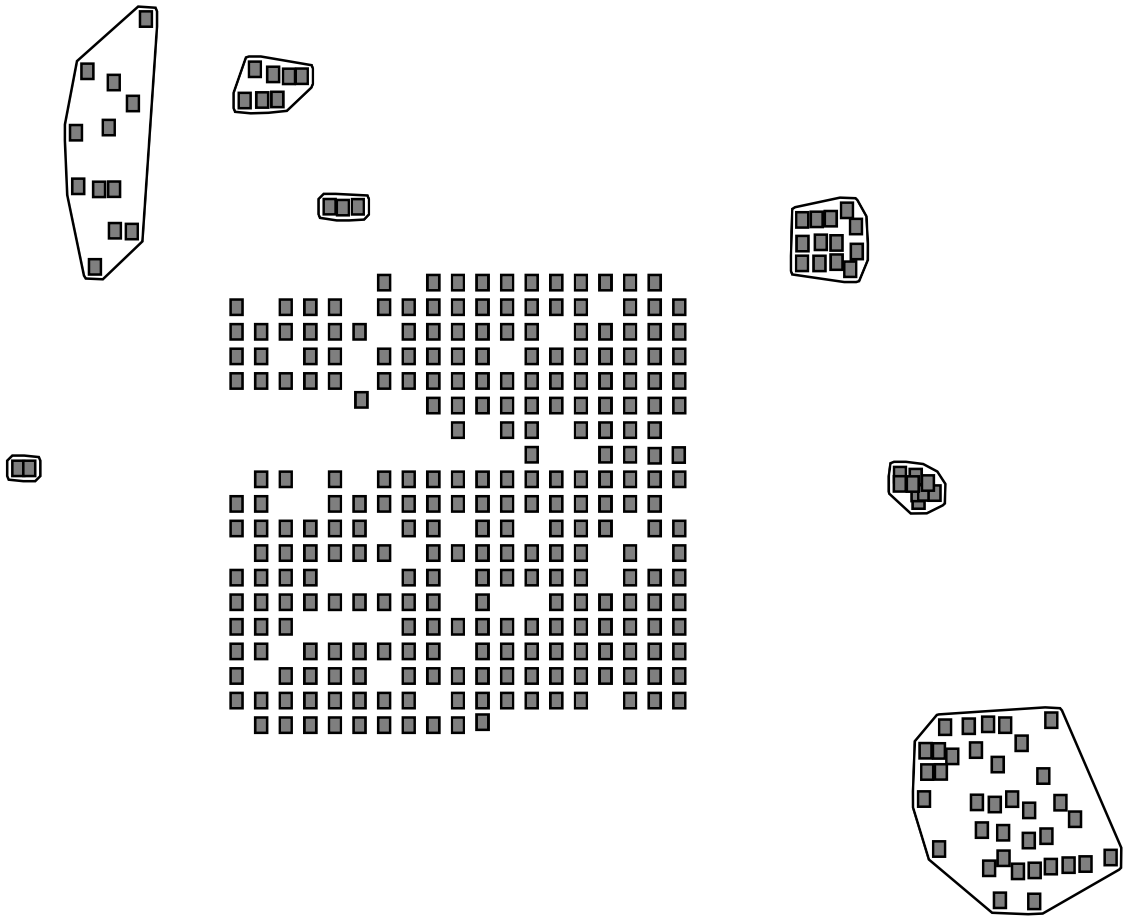


Figure 4.7: Workspace at session A1 end

Contrast this strategy with **A**'s earlier randomization. With randomized selection, each extraction has essentially four steps:

1. Select an item at random;[4]
2. Identify it, often by zooming in;
3. Decide its category;
4. Move it to the appropriate pile. Repeat from step 1.

Batch consolidation has similar steps, but in a different order:

[4]Selection is never truly random; here it means only that **A** deliberately avoids selecting neighbors sequentially.

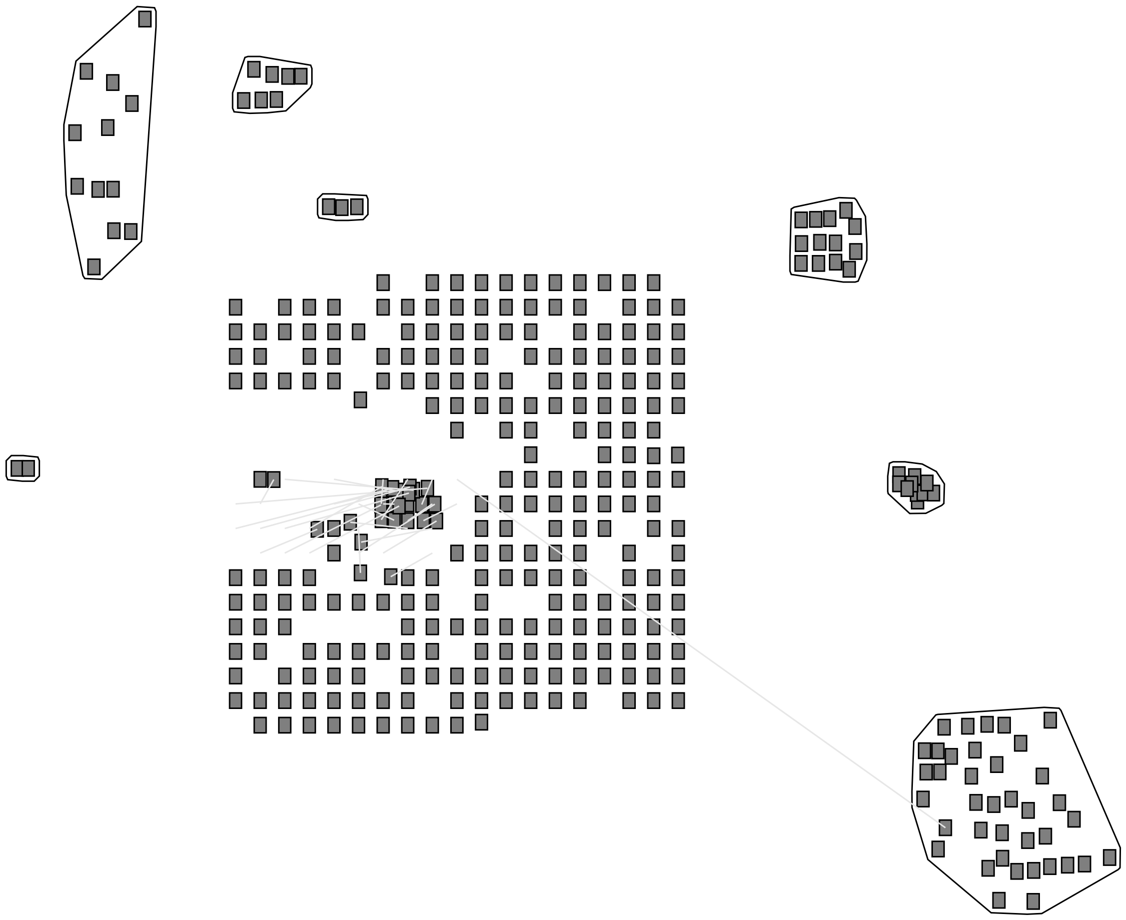


Figure 4.8: Consolidating a local batch of related items *(A2:09:47)*

1. Decide on a category;

2. Select an item locally;

3. Identify it;

4. Verify whether it belongs in the category;

5. Move it locally. Repeat from step 2.

The consolidation strategy has two advantages. First, as **A** suggests, it saves mechanical effort by taking long-distance travel out of the loop, deferring it to the end and replacing that step with an easier, local action. Second, it saves cognitive effort by deciding a category initially and substituting that step with an easier verification.

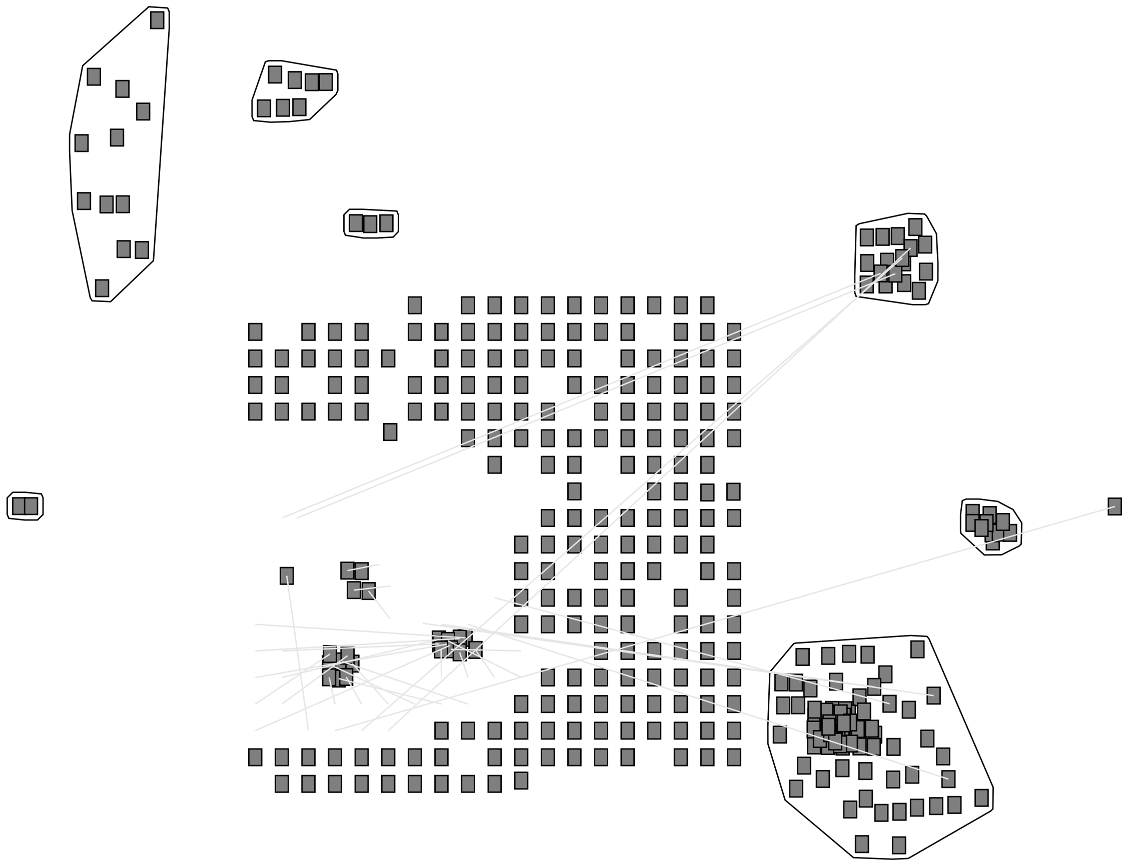


Figure 4.9: More local consolidation. The batch in Figure 4.8 has been moved to *HCI*.

Of course, this is exactly the opposite of **A**'s earlier intent, expressed in session 1, to increase cognitive effort and rethink each item.

What is the point of moving the item at all, since it will be moved again? That accomplishes two things:

1. It marks the item as a batch member, so that the validation effort is not lost; that is, members are given visual contiguity.
2. It gives the batch physical contiguity, making it easier to select and cohere the items for the long-distance move.

But these functions could be accomplished at least as well by stacking or partially occluding the items. Instead, **A** merely abuts them to preserve their visibility. This potentially serves two additional purposes:

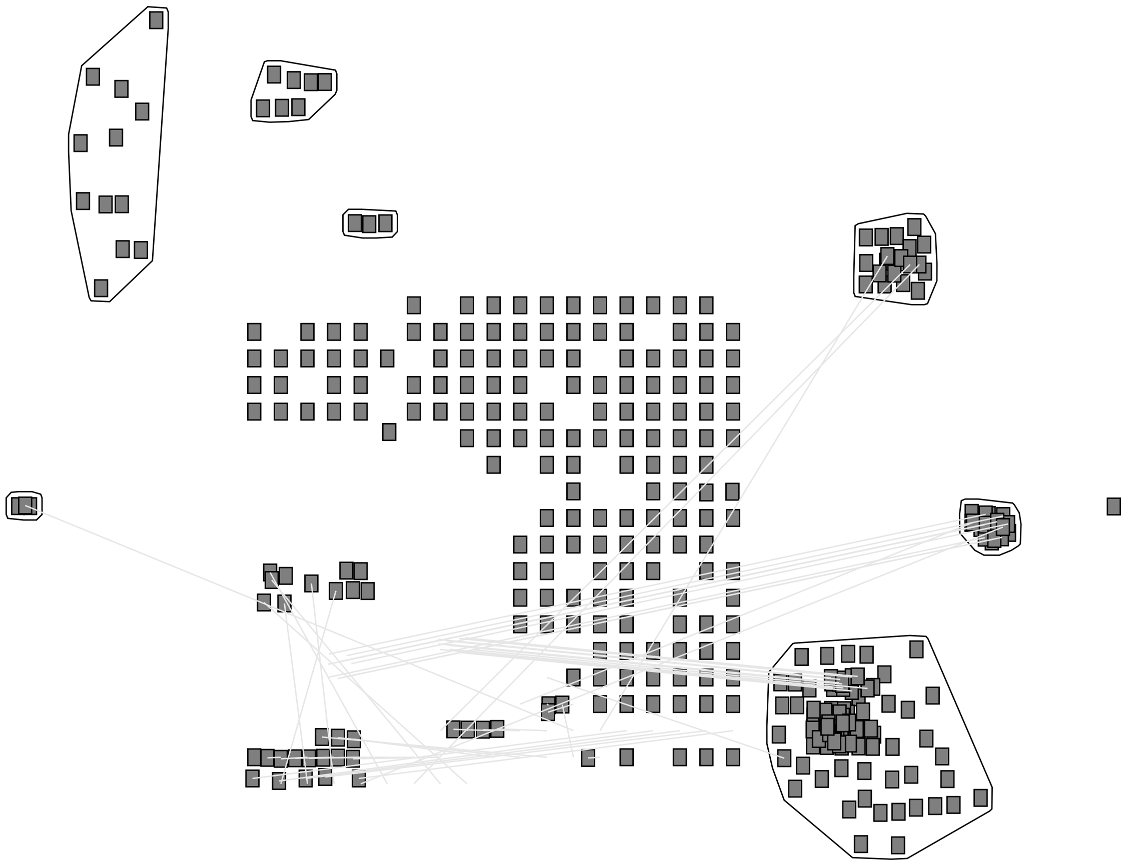


Figure 4.10: Still more local consolidation *(A2:37:25)*

3. unstacked batch members have greater spatial freedom and can therefore express information by their position, and

4. their visibility may serve as a reminded for possible refactoring.

It appears however that **A** does not employ these possibilities for the batches shown in Figures 4.8–4.10. He never removes members or subdivides the batches, and offers no sign that his rapid abutments are thoughtfully placed. In any case, any ordering within a batch is lost when merged with the destination pile, as we'll see shortly.

Potential for Overlap A variant of this strategy could have been even more efficient. Because the destination piles are wrapped in explicit containers, they are cohesive and insulating. So **A** could have temporarily moved them to the work area, placed them safely atop the grid, and loaded selections into them directly, instead of batching.

One potential deterrent is that Dynapad's containers are slightly transparent and underlying items can still be seen faintly through the container body, amidst the members on top.

I demonstrated the technique of bring the piles close, but **A** didn't adopt it, seemingly because he didn't like the visual interference of the underlying items. But ironically, his strategy produced as much disorder in the piles as if he had tossed items in blindly.

Modal "Tossing" We might describe this consolidation strategy as "modal": in choosing a category initially, **A** effectively enters a "mode" for that category, oriented to a particular destination pile. We could imagine Dynapad supporting that mode directly: if the user specifies a target pile initially, then approved selections could be easily "tossed" into it at a distance, avoiding the effort of both long-distance hauling and local consolidation.

But this misses some advantages of **A**'s strategy. First, it eliminates functions 3) and 4) above, since the tossed items presumably would no longer be visible and could not be expressively placed upon "landing". Second, it would deter opportunistic mode-switching: starting an additional batch in response to an unexpected item. **A** *does* do this; as one example, Figure 4.9 shows **A** assembling two batches concurrently.

In summary, a modal strategy like **A**'s batching is a gamble, compressing activity by betting on its redundancy. It potentially saves effort by consolidating a repeated step into one collective initial or final step, but it presumes the activity will be relatively homogeneous. We might interpret **A**'s batching as a risk-reducing compromise, saving only minimal effort with each item but reducing the cost of potential interruptions.

Protecting Expressive Placement. Figure 4.11 shows a close-up of batched items at the left of Figure 4.10. A single item lies midway between two piles, placed there because it belongs to both categories. As **A** says:

35 **A:** Hmmm... it could be in between. It's kind of miscellaneous [gestures to left clump] but also kinda has to do with money [gestures to right].

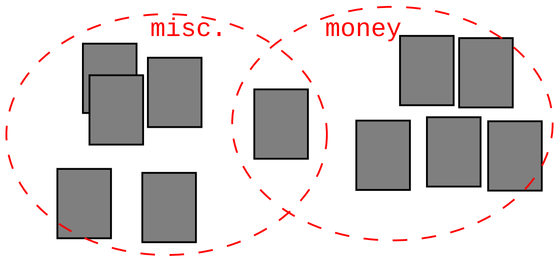


Figure 4.11: Closeup of working piles. Note the single document between the piles; it was intentionally placed there as a tentative member of both categories (pile "spanning")

Unfortunately, this expressive placement is lost later on. Because the isolated item does not cohere to either pile, it is left behind when the right-hand pile *money* is moved to related items (Figure 4.12). This is essentially the same problem as with implicit groups: unlike explicit clumps, implicit groups are unprotected with any "coherence" physics.

How might this be mitigated? One solution of course is for the user to copy and multi-pile the shared item. But the ability to "span" piles is one of their advantages, and ideally Dynapad could preserve it.

Partial membership: Viscosity and Graded brushing With explicit clumps, classifying members automatically gives them cohesion; their relationship to the group is "protected". So why shouldn't partial members receive partial protection? We could imagine a "viscosity" surrounding piles: nearby objects would partially stick to their moving host, perhaps following it a short distance before breaking off, giving the user a chance either to keep or reject it.

Another solution might be "graded" rather than binary brushing: partial or tentative relations could be remembered and highlighted (perhaps more lightly than a full relation) at a distance if near-relations are inadvertently separated.

Pile Overloading. Figure 4.12 shows the result after moving several batches into the "course" piles on the right: they are all very dense, overlapping, and seemingly chaotic. How did they become so overloaded? This is perhaps the central phenomenon which determines the course of **A** work.

One of the primary causes is that when **A** adds a batch to a pile (such as *non-*

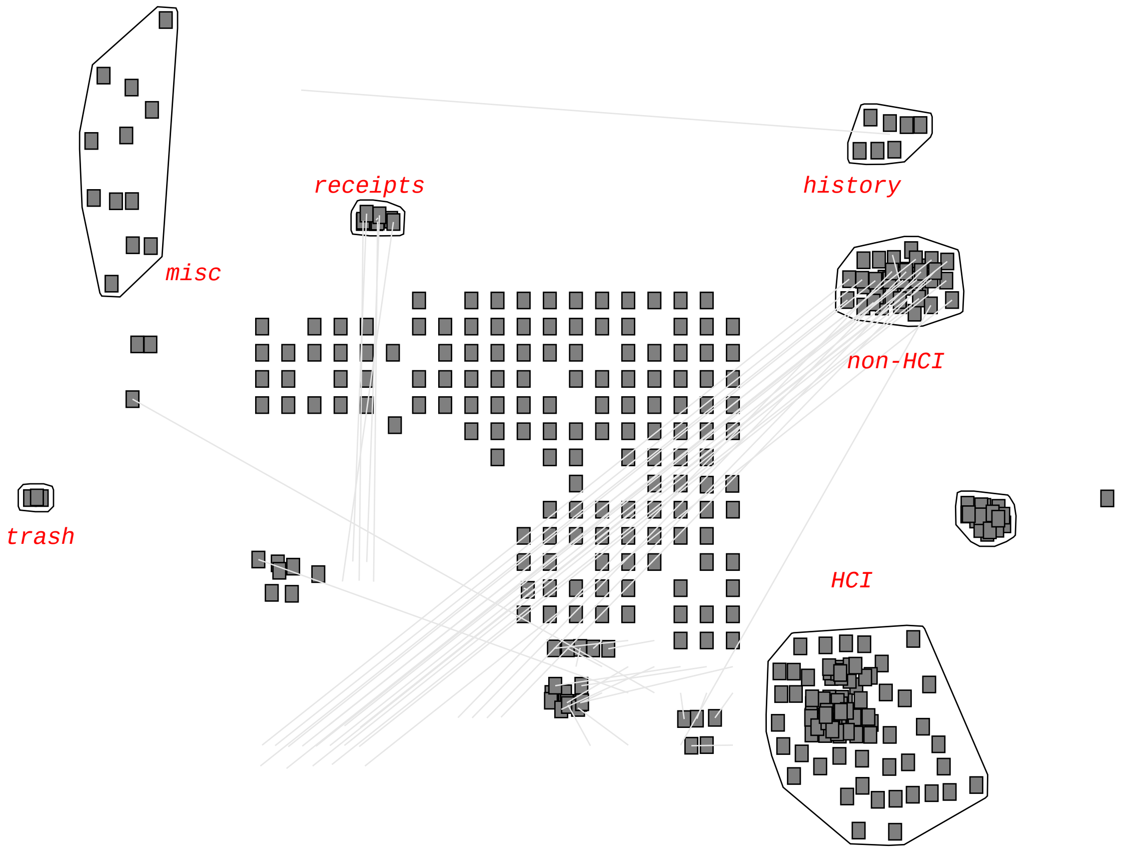


Figure 4.12: Moving batches into piles: SW batch has been dumped into *non-HCI* pile, losing information

HCI, as in Figure 4.12), he does so in way which occludes the items and any organization already there. **A** himself notices the problem:

36 **A:** You know... how these <*local batch*> are piled together? It'd be interesting if it noted I did that.

37 **I:** What do you think it should do at that point?

38 **A:** Like... I'm gonna probably take this, right... I'm gonna circle it [lassos it], and then I'm gonna toss it in [moves set] my *neuro* pile... (that is, my *non-HCI-courses* pile) [drops it]. So now it's in there, but now it's lost association with each other.

39 **I:** (Explains ways to do that, with a sub-pile or careful placement.)

Apparently, **A** has a misconception about the clump's behavior. After a similar action, he comments:

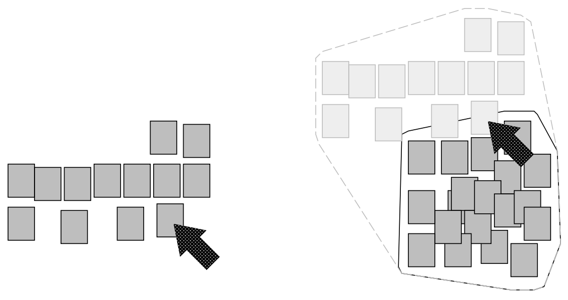
40 **A:** [Drops new batch into clump center; its edge doesn't change] Doesn't the pile grow?

41 **I:** (Explains)

42 **A:** But the thing is, I know if I used, like, a lens on it, it would do something, right? It would force it...

A clump doesn't expand merely from density, but when outlying items stretch its border. This can happen when an existing member is dragged outward, a new item is added at the edge, or a widely-spaced[5] batch is dropped with the pointer inside the boundary. In the last case, the clump's *greed* (see 3.6.4) takes even the members that would otherwise fall outside.

Therefore the *ideal* way to add a batch without occluding items is to grip it at its edge and then drop that point at the clump's edge, as shown in Figure 4.13(a). However, gripping the correct edge requires *anticipating where in the clump the batch will be added*, which is prohibitively difficult at a distance. Without much experience, **A**'s reasonable intuition is to grab the center of the batch and drop it in the center of its destination, as in Figure 4.13(b).


(a) Ideal: grip and drop at edge to avoid occlusion.

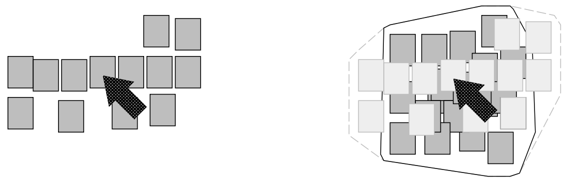

(b) Temptation is to grip and drop at center.

Figure 4.13: Adding a group to a pile can be done badly.

[5]relative to the clump

Clumps have no Memory. The above problem is compounded by the fact that clumps do not remember the order or co-arrival of its members, as **A** noted above. He adds later:

43 **A:** I should've done this <*making local clumps*> before...

44 **A:** [Finding another <*174*> item:] Oh yeah, I remember I had a 174 pile, but I just tossed it in there <*non-HCI*> without making it a pile.

45 **A:** I'm realizing I should've made probably smaller piles.

By now **A** has recognized that, because his initial categories are too broad and his batches have been mixed together, he has lost information in his piles' internal structure.

The Ontogeny of Overloaded Piles. **A**'s piles descend into chaos in a consistent pattern, with two phases:

1. Before wrapping the piles as explicit clumps, **A** put new items adjacent to old ones, growing their implicit boundary outward (Figures 4.6–4.7).
2. After being explicitly wrapped, because of the problems above, the piles mostly stopped expanding and instead became more dense. Items became occluded, informative whitespace vanished, clarity decayed, and chaos bloomed.

This seems to be an extreme case of an overload cycle (see Figure 2.12) within individual piles: as clutter increases, clarity drops, clean-up is difficult and deferred, so **A** doesn't even bother trying to maintain order. He says later (session 3) of his *HCI* snarl:

46 **A:** So that <*document*> could go... in this big pile <*HCI*> [moves document into pile] since it's all unsorted.

47 **I:** What do you mean, since it's all unsorted?

48 **A:** Oh, this is all unsorted, and then later on, like in the second session I figured out, hey, maybe I should sort... sub-sort piles.

49 **I:** (Clarify: since its unsorted, therefore?...)

50 **A:** It means I'll put it in here, then later on I'll sub-sort... I'll re-sort this <*HCI*> pile itself.

51 **I:** Is that a sense of, oh well, it's already so ... tangled...

52 **A:** Yeah... I'll deal with it later.

53 **I:** ... it's hopeless, I might as well just throw this in the batch?

54 **A:** Yeah. It's just a big bag of... It's kind of organized, cause it's HCI-related, but then it's also not— see, here's another HCI thing [picks it up]. I might as well throw that there too.

Why Piles Get Messy: Too Much Rope. By now we've considered several aspects of Dynapad's design which contribute to the messiness of piles. Let's review them briefly here.

Over-broad categories Categories like "HCI" and "non-HCI" are disasters waiting to happen. The category of "HCI" was articulated *with the very first member* **A** *found* before any opportunity to reflect on the distinctions that would be effective. The three largest piles, the broadest categories, were the first formed.

Inducement to batching **A** began batching to reduce the cost of repeatedly transporting items to piles out beyond the edge of the grid. They formed there because the original grid left no internal space, and they remained there (in part) because of **A**'s reluctance to move overlap them.

In addition, the partial organization in the initial grid, combined with **A**'s broad categories, made it easy to condense (and conflate) many items at once.

Misleading physics metaphor Clumps have straightforward behavior, but that modest automation *suggests* a metaphor with more thorough automation: that clumps expand under "pressure" to reduce density. An unwrapped, implicit pile creates no such expectation, so **A** invests manual effort to keep members exposed. Ironically then, modest physics may be worse than none.

Misaligned cost-structure The easiest, more obvious affordance for adding items to a pile, centered gripping and dropping, is not the most effective. The more effective interaction is physically more difficult and counter-intuitive.

No memory The chronology and coordination with which items are added to a pile could be harnessed to enrich their substructure, but the information is ignored and forgotten. The problem is compounded with batching and center-dropping.

Poor harvesting of extant structure One last influence we've not discussed, but we'll see ahead. Even when informative substructure is available in a pile (i.e. directory brushing), it's difficult to leverage that information in the primary, spatial organization. We'll see a concrete example later in session A5.

Belated Wisdom. By the end of session 2, **A** has learned the lessons above and begun to work with smaller, more specific piles. Figure 4.14 shows his workspace at the end of session 2, annotated with **A**'s implied pile names.

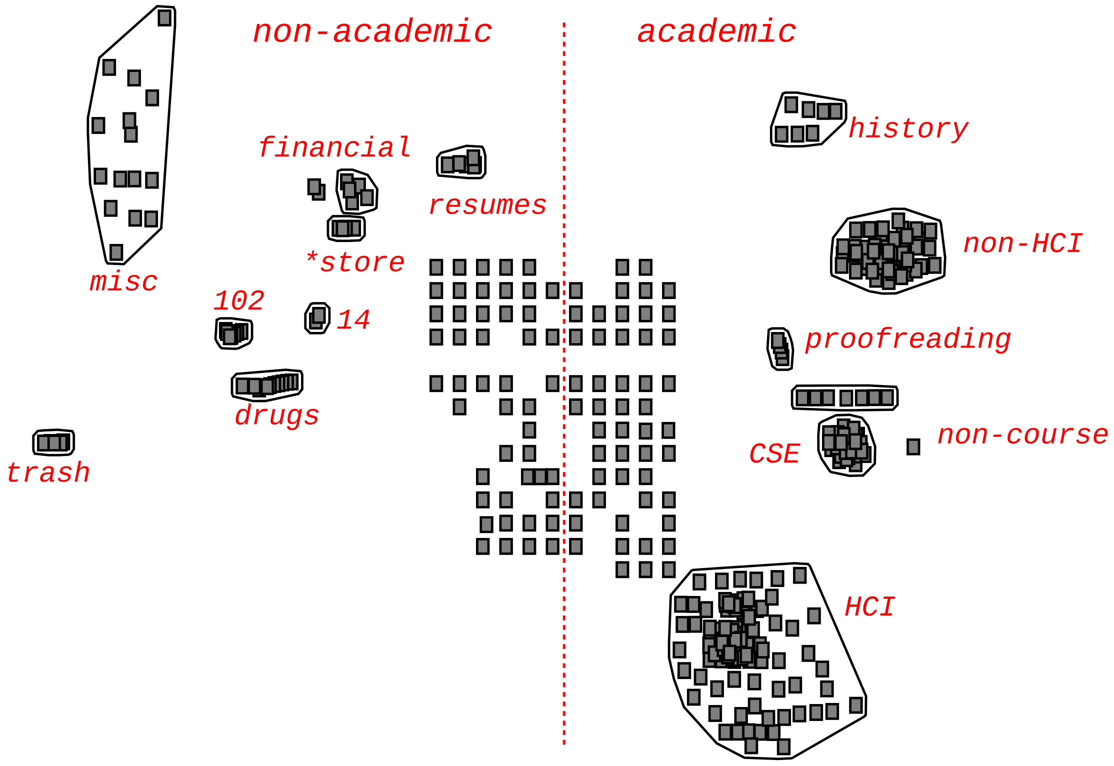


Figure 4.14: Workspace at session A2 end *(A2:68:07)*

Session A3

Improvised Labeling. While de-occluding items in his *Proofreading* pile, **A** has an idea:

55 **A:** Maybe I should spell something... I could make a P... (which) stands for "Proofreading". [rearranges into a crude P] Eh. We'll work on it.

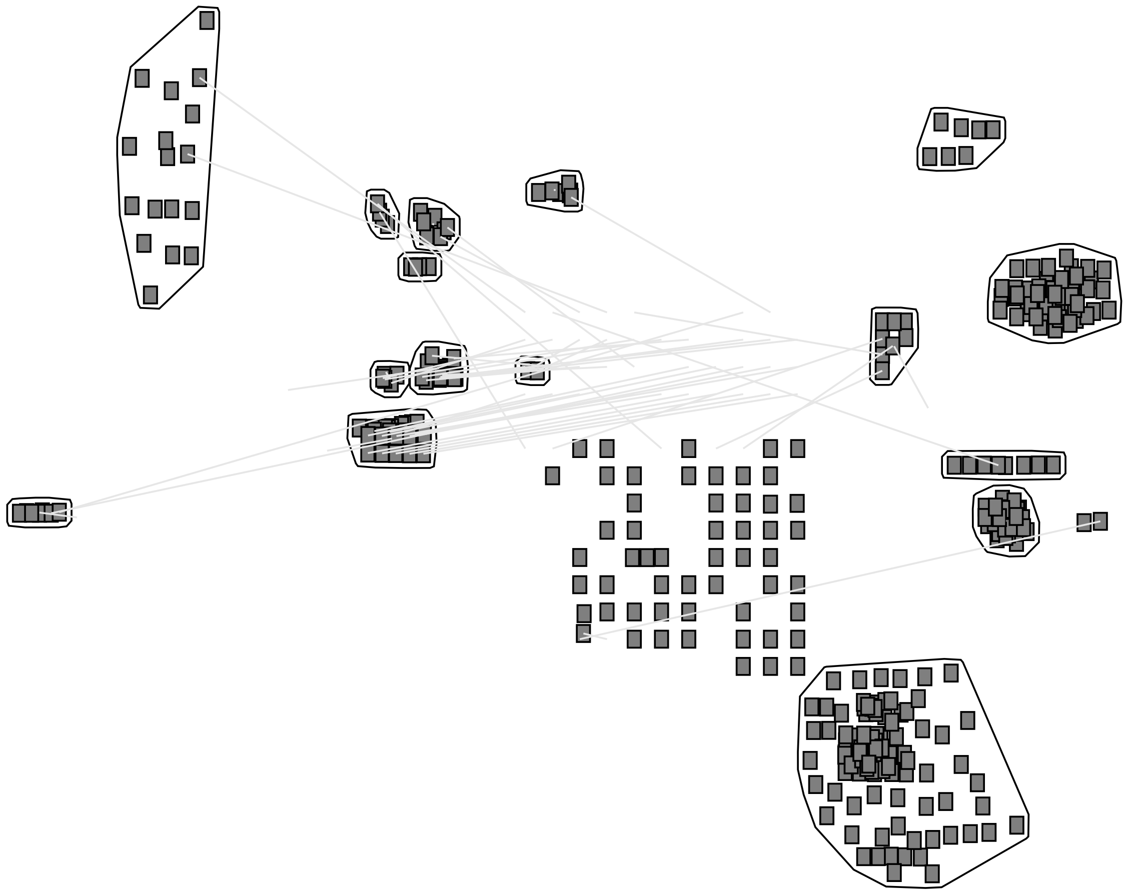


Figure 4.15: After improvised “P” label *(A3:27:18)*

56 **I:** So... if you had the capacity to label these piles with some chunks of text, that would be helpful at this point? You would have used that instead of the P?

57 **A:** Yeah. I think... because... there are piles, and unless I explicitly remember them by location... and size, kind of like... that’s <*Misc*> kind of distinct, cause I made a big pile, but it’s not really... condensed. You know, all the stuff is freely distributed; there’s like a lot of space; it’s like, kind of ugly looking, so I was like, OK, that’s the miscellaneous one. [Points to HCI pile:] And this thing’s... really condensed, and it’s the biggest one, so I remember, OK, this is all my HCI stuff. But then, when you get into these smaller piles... [gestures at P and neighbors] I have to like zoom back in [zooms in] to see the pictures and I understand what it is again...

58 **A:** So now that I see that P, I’m never gonna ... be wondering what that is again.

59 **I:** Unless of course you forget what P stands for.

A's attitude about labels has changed from earlier. By now he's learned the aggregate appearance and positions of the dominant piles, and still makes use of zooming glances to remind himself of others. But with more piles, he has a greater demand for explicit labels, and he's willing to invest effort in giving piles distinctive aggregation to compensate.

By the end of session 3 (Figure 4.16), he adds an R ("resumés"), begins an S ("statistics"), and begins a "120" (for course 120). In forming a separate 1 and 2 for this label, he justifies a distinction that he likely would have ignored otherwise:

60 **A:** And so everything in this 1, I could keep it as... things I wrote. And everything in the 2 would be... things that were written. Not by me, but by anyone else.

His labels, contrived for another purpose, create a top-down pressure to subcategorize which interacts with the bottom-up diversity of individual items.

Still later, in session 4, **A** revives the 120 label, but it becomes "110" instead: the compression of the boundary, combined with effort of adjusting many items, made it too difficult to space the digits, and there was inadequate room for a "2" between the "1" and "0". The end result is a compromise of bottom-up division into 3 subcategories (homework, lectures, and readings), and top-down pressure of contrived labeling.

Use of Brushing (scene 2); Fighting the Monster. Finally, near the end of session 3, **A** begins to clean up the *HCI* pile, inspired by the ambient brushing signal:

61 **A:** What happens if what I'm highlighting right now [brushes a 120 doc]...[sees highlighted document in adjacent *HCI* pile, extracts it and zooms in] It (is) a slide!...

62 **I:** So what's your inquiry here?

63 **A:** ...and it's 120. I can start cleaning up... this <*HCI*> pile.

64 **I:** So you're suggesting that all the things which light up when you select these 120s [points to 120 pile] are also 120 (here) [points to *HCI*], therefore they're easy to pick out [points back to 120 pile].

65 **A:** Well, they don't have to be, because... since I sorted things chronologically, it could just be stuff that ... was during that time period, but not necessarily—

66 **I:** But that's what you meant?

67 **A:** Yeah.

A has discovered he can use brushing to recover some of the information lost in his haphazard conflation of *HCI.* Neither all the brushed items nor all the HCI items are 120s, but the intersection of the two larger groups, one spatially contiguous and one distributed, yields what he seeks. He begins to collect and condense those within the HCI pile, enriching its internal organization.

68 **I:** So, at this moment, you're actually going through this clog [touches HCI pile] and you're making sort of ...[gestures around pile perimeter] subregions that correspond to categories?

69 **A:** Yeah.

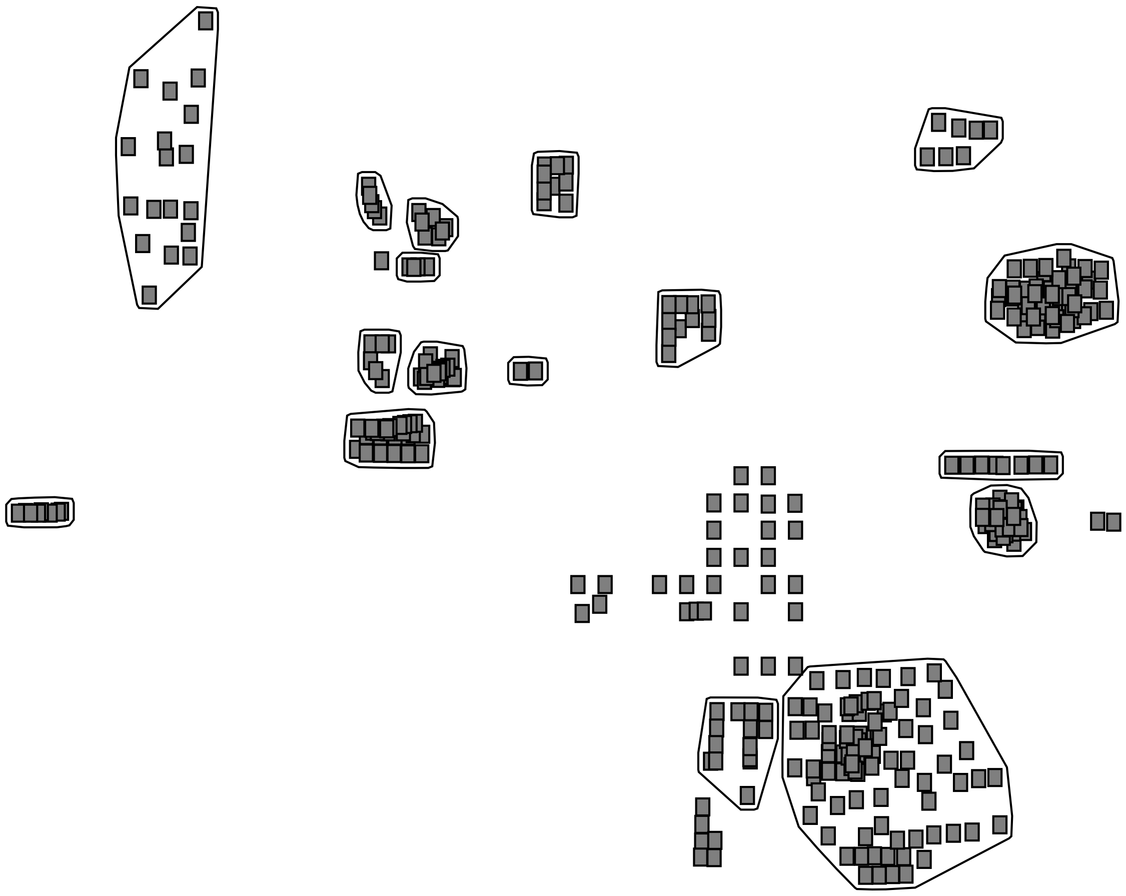


Figure 4.16: Workspace at session A3 end *(A3:56:56)*

Session A4

Early in session 4, two weeks later, **A** categorizes the last of his original reservoir. His original reference landscape, the grid, has now been completely replaced with a landscape of piles. **A** resumes the second phase of organization, cleaning and subdividing his largest piles.

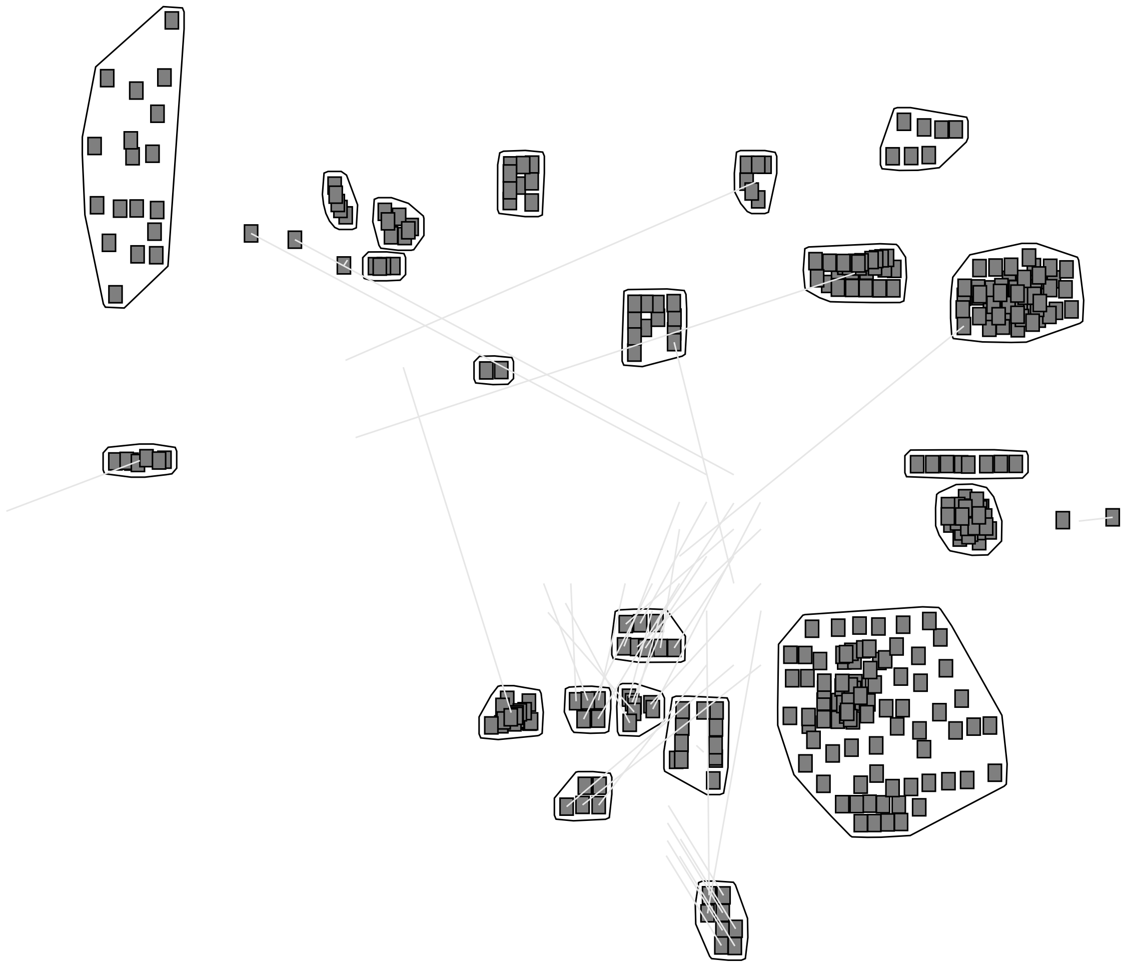


Figure 4.17: Reservoir depleted, piles positioned; time to begin re-ordering chaos piles.

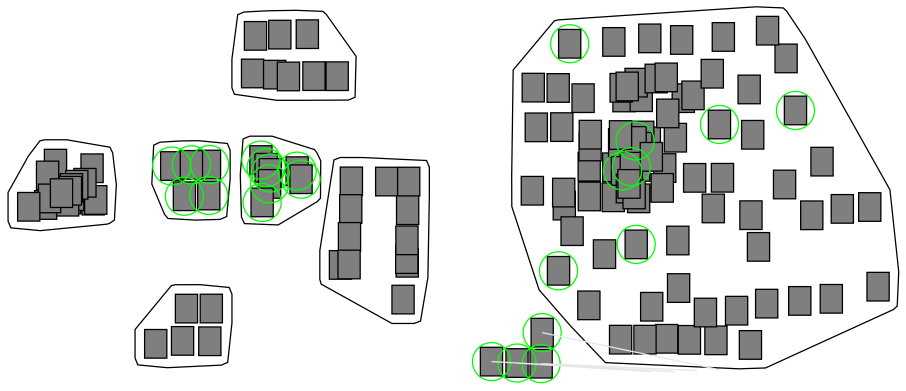


Figure 4.18: 4 items extracted from HCI pile; one is brushed, highlighting others.

Use of Brushing (scene 3). As before, **A** uses brushing to subdivide *HCI*, first extracting four related items (Figure 4.18):

70 **A:** Ethnography project [pulls one]... ethnography project [pulls another]...[Collects and abuts 4, then begins looking for others by brushing one] The highlighting is great! [Find another related doc, pulls it]

71 **I:** So, you're looking for things in the same (academic) quarter?... What's your interpretation of what gets highlighted when you mouse-over this <*source*> guy?

72 **A:** Well... the low level thing is I know it's based by quarter, but I'm not thinking that, I'm thinking it's just something that has the potential to be—

73 **I:** Show me something that might be relevant, to help direct my attention?

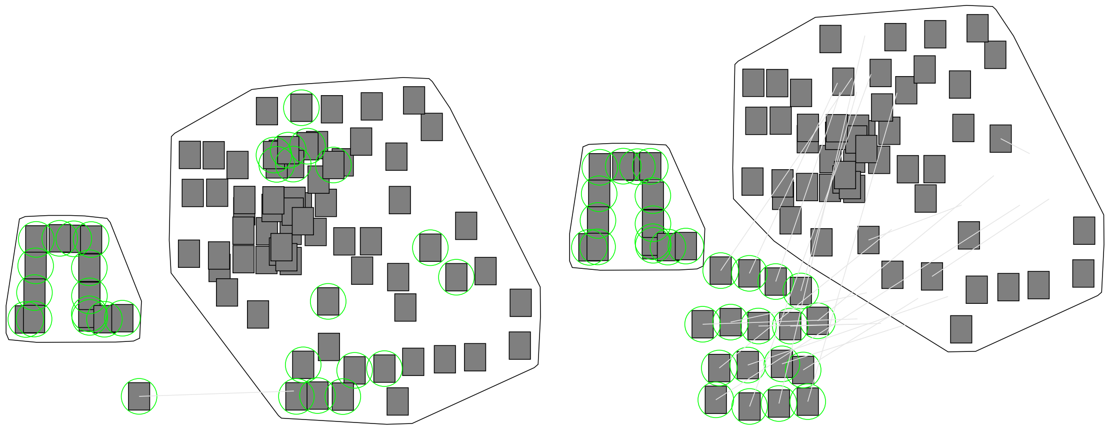
74 **A:** Yes. Cause it's a huge pile, and it's like— see, now I'm gonna take it out, I'm gonna look at it, and if it's part of this group <*ethno*> then it's (in); otherwise I get to toss it back [gestures back to HCI].

A is using the highlighting to direct his attention to possible selections within HCI. A minute later, he uses it instead to spot likely *destinations* for those selections:

75 **A:** [Holding an *HCI* doc:] Well, since this <*two nearby piles*> is highlighted, I know it's one of these two piles.

Use of Brushing (scene 4). Earlier, **A** used brushing to consolidate manually a highlighted subset of a pile. Now he expresses a wish to automate that procedure. Unfortunately, no such operation is possible. Dynapad can easily select all brushed item or all items gathered locally, but not the *intersection* of those sets.

This is partly a consequence of Dynapad's overloaded selection mechanism (see 3.2.2): by combining incremental selection with inversion, it cannot emulate set *subtraction* to achieve an intersection.



(a) Preparing to visit brushed set within pile *(A4:42:36)*

(b) Brushed set extracted and serialized *(A4:45:43)*

Figure 4.19: Extracting a brushed set

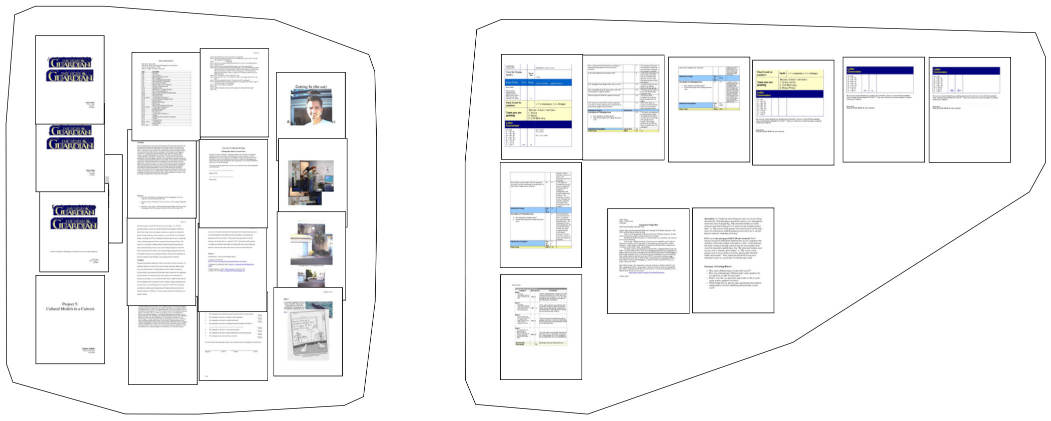
A is forced, once again, to process the pile's brushed items one by one. This is a serial process, and to simplify it, **A** first extracts and serializes the items into a grid (Figure 4.19).

This is an example of the final contributor to pile chaos mentioned earlier (page 185): the difficulty of leveraging available information for spatial organization.

Ironically, the condensed serialization that **A** creates here is exactly the kind of low-level automation that region-tools were intended to support. **A** might have put those selections into a grid-tray to save some effort, but even that would have been only one component of the task.

Serial Visitation. The real goal of Figure 4.19 isn't to create a grid or even select the brushed items, but to *visit* the items efficiently and exhaustively. What Dynapad could better offer, for this and many other such examples, is a *serialized visitation* of a set of items, regardless of their arrangement.

Enriching Aggregation. **A**'s letter-labels were one example of manually-enriched aggregation. Late in session 4, **A** creates other expressive arrangements, less symbolic but equally distinctive. For example, the pile of Figure 4.20(a):



(a) Grouped logo, text, and images (b) Images outside, "meat" inside

Figure 4.20: Two clumps arranged for aggregate distinctiveness

76 **A:** Hey, let's put all these things in a row. [Begins aligning documents with dominant logo image].

77 **I:** Why?

78 **A:** It'll totally direct my attention to what it is, cause the logo is unique, and then I'll put the pictures on the other side [begins aligning others w. dominant image]. And the text in the middle... [arranges non-image document between those rows]. Because, I don't know, that's like the important part, but this is just like... [gestures around perimeter]... (making) a little cute border.

79 **I:** So... you're giving this pile visually distinctive character...

80 **A:** Yes.

81 **I:** ...by... appealing to the visual coherence of the large images and the cover page?

82 **A:** Yes.

83 **I:** Is that close?

84 **A:** That's it. On the spot.

Later, **A** applies a similar pattern to different pile, in Figure 4.20(b):

85 **A:** [arranges graphical items on border]

86 **I:** Why (a) border?

87 **A:** Well, you see how they're like... the blue... it's just to distinguish (them)...

88 **I:** You want the visual... stuff on the outside, so that you know where to look for it?

89 **A:** And then the "meat" <*text documents*> on the inside.

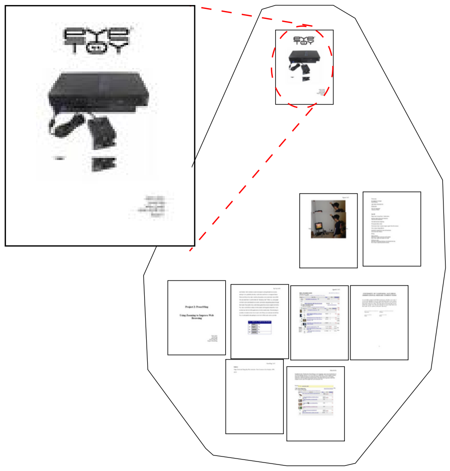


Figure 4.21: The "eye" of the pyramid

And finally, an arrangement of an "EyeToy"-related pile (Figure 4.21) with a clever mnemonic:

90 **I:** So when you put this <*EyeToy doc*> into that pile just now, how did you know which pile it was?

91 **A:** Well— originally when I made this [circles pile]... structure...this [circles top-most doc] is my distinguishing object in this pile—

92 **I:** The peak of the pyramid.

93 **A:** Yes. And then– you know how the peak of a pyramid has an eye?

94 **I:** Ah....

95 **A:** On the back of the dollar bill?

96 **I:** I get it. So you could actually tell that that was the EyeToy picture at the small scale?

97 **A:** That, so... yeah...because when I did this [moves second item to peak] before, it kind of ruined it and made it look like every other pile [moves item back below peak]. But the fact that... this is this <*the eye*> and this is a pyramid-looking thing is like... symbolic.

98 **I:** So it's not just that you're seeing the picture; you're seeing the whole shape of the pile.

99 **A:** Yes.

All of these enrichments are possible only because Dynapad's piles are "open": items' spatial freedom allows expressiveness, enhanced by their aggregate visibility.

Figure 4.22 depicts the end state of session A4.

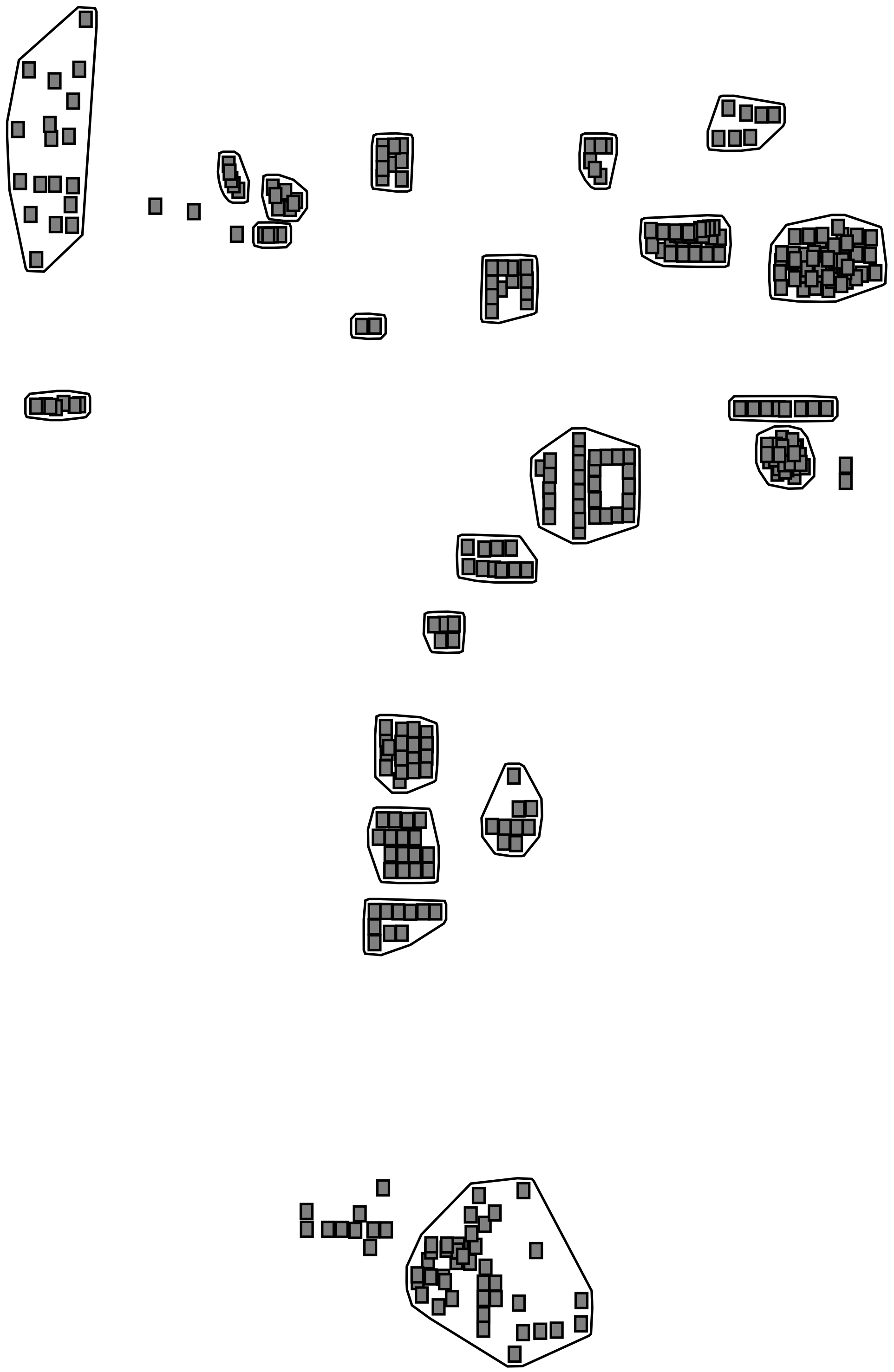


Figure 4.22: Workspace at session A4 end *(A4:72:43)*

Session A5

Session A5 took place more than 3 months after session 4. To reorient himself, **A** watched the last few minutes of the video of session 4, re-establishing his unfinished goal of cleaning up his *HCI* pile. But his review showed only the very bottom region of his workspace; any understanding he has now of other piles has been remembered from months before.

Labeling vs. Implicit Organization. By session 5, Dynapad's features included the slow-zooming labels on piles (see 3.4.3). Shortly after the start of session 5, **I** introduced **A** to these labels, and he chose to label immediately all piles.

A moved many of the piles once he had labeled them. His motivation is unclear; it may be a continuation of his established "process and set aside" procedure. In some cases, labeling piles causes **A** to reconsider their relationship. For example, after **A** has labeled a pile "ethnography", he considers another pile for which the same title could be appropriate:

100 **A:** Ah, this is 102A. But it's still ethnography. Why did I separate these? I think maybe because these <*unlabeled*> were bigger projects, or... this is all the UCSD Guardian, pretty much.

101 **I:** So... is this a subcategory?

102 **A:** Of ethnography, yeah. What would I do if I wanted to make it— can you put a pile in a pile?

103 **I:** Yep.

104 **A:** OK, first I want to give it a title [titles it "Guardian"; then places it in *ethnography* and adjusts to de-occlude].

Later, *ethnography* also subsumes an "assignments" pile.

Spatial Memory. Regardless of **A**'s motivation for moving labeled piles, it seems clear that the labels reduce the primary *deterrent* for doing so: the danger of disrupting an implicit and spatially-based organization. Having labels reduces **A**'s need to identify piles by shape and position. As he observes:

105 **A:** See, now you don't really need to spread things out as much, since now you have titles. Things can be in closer proximity, and you don't have to be like, OK,

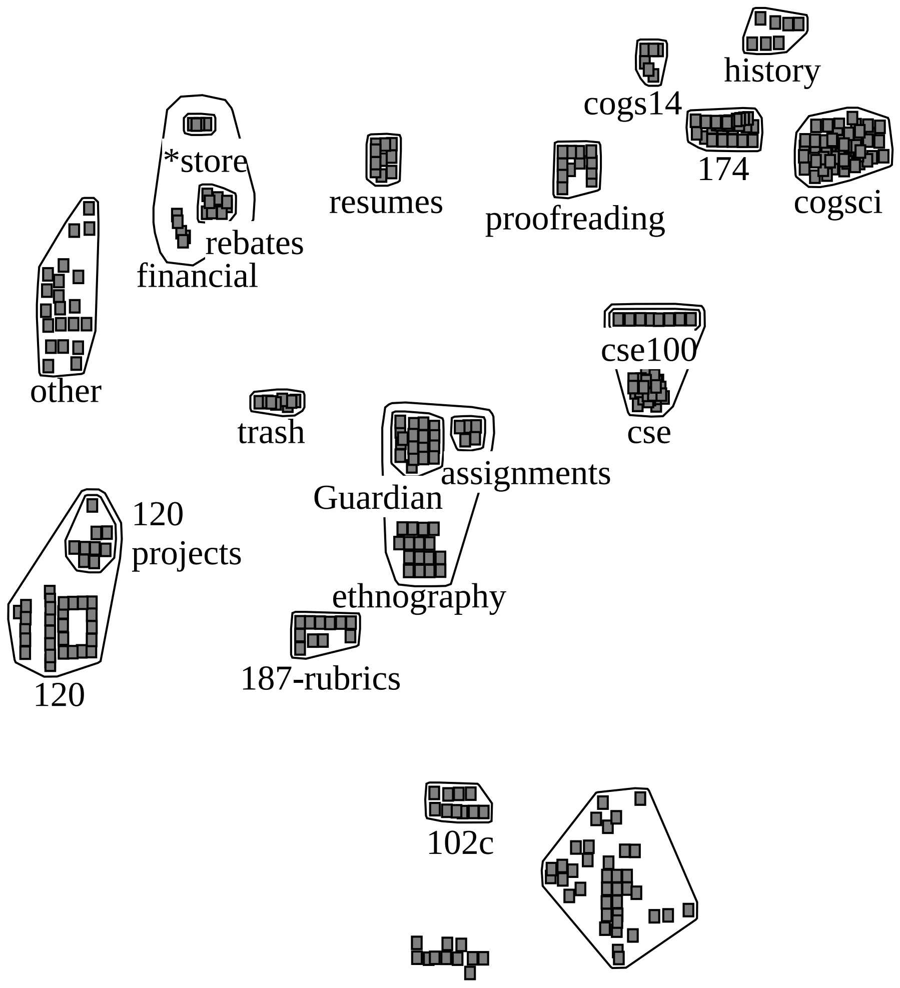


Figure 4.23: Workspace fully labeled *(A5:37:22)*. For clarity, certain labels in this figure are displaced from their normal position below their piles.

the northeast [circles NE piles, as yet unlabeled/unexamined in A5] is this kind of area, and the south side [circles there] is... HCI stuff. Now... I can reference it quickly.

106 **I:** Were you using those (just) as examples, or do you remember, is it the case that in this (workspace) northeast is...

107 **A:** Yeah, I remember... this area [circles northwest] was more miscellaneous, this [circles south] is more HCI stuff, [circles northeast] these were cogsci[6] classes but not HCI... I remember I had a big pile [zooms straight into non-HCI], see, this is all 107.[7]

[6]Cognitive Science department

[7]Actually **A** is mistaken; this pile now includes other courses as well.

108 **I:** ...that's the first time you zoomed into that, and you couldn't actually see... you couldn't confirm before, but sure enough, it's what you expected... Do you have some sense of what it is that let you remember what to expect there?

109 **A:** It's because I explicitly made this area... I remember I made the area for that reason. And so... like you know, everything that's... it's based on area, and I did it because we didn't have the titles before, so... you just try to make zones.... And I guess I probably had three zones, and what used to be in the middle is what was being sorted. But now everything is sorted, so we just have a bunch of zones that I have to rediscover.

And later, when all piles have been labeled:

110 **A:** See now how everything's clustered? It's no longer a question mark!

111 **I:** So now you're relying more on the labels than on the spatial position?

112 **A:** Yeah.

Final Reflections. **A** gradually groups his piles in several levels, culminating in the state shown in Figure 4.24. The conversation turns to a comparison of exploring an organization in Dynapad and in a file system.

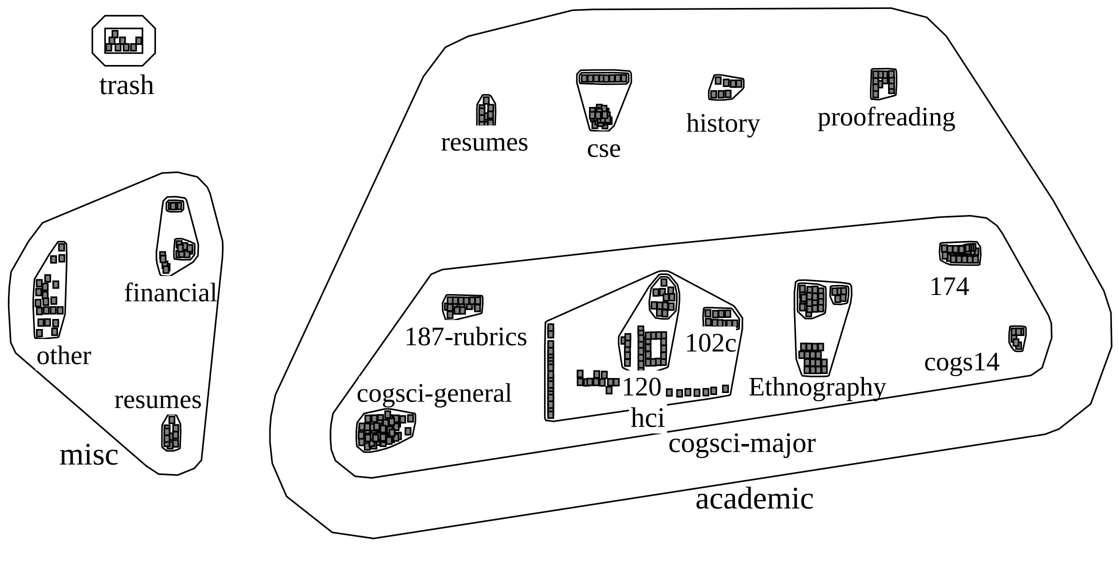


Figure 4.24: Final workspace state *(A5:110:25)*

113 **A:** Well, see, here... I'm seeing... this whole thing [gestures over entire workspace] you can pretend is a directory... of my documents. Here [gestures to *academic*] you've got the folder called "cogsci-major"...[points within it] here I'm seeing a

subdirectory... [points with multiple fingers to piles still deeper:] within that subdirectory I'm seeing their subdirectories. You could never do that in the Windows structure; you can only see one... directory at a time. ...Like, I'm seeing every level right now.

In short, visibility improves the clarity of the organization. **A** continues a minute later:

114 **A:** So that's why... you're not as trapped; you can see, [gestures from *academic* across to *misc*] "Oh yeah, that could go in that folder", especially (when) you can give it titles. And then you can just compare this [touches *academic* pile]: "Well, yeah, resumes could go in both worlds". Whereas when you're in a directory, it's... you don't see anything (other than) what's in there... unless you open two windows. But then that's not a natural way of exploring.... cause it's not spatial.

115 **I:** (A collapsable outline view, although one-dimensional, tries to accomplish the same thing, to show several levels and their structure at once. So what's still different about that? What does our two dimensions buy us?)

116 **A:** It's easier for the eye. ... Like, a tree structure isn't pretty natural, you know, but... when... you arrange things on this table, that's a pretty simple, natural... thing for you. But, you know, you wouldn't ... naturally create a tree structure.

117 **I:** Well...

118 **A:** This is more primitive in that sense. Like, it's more... primitive in a good way.

119 **I:** Primitive? More, somehow...

120 **A:** The use of space.

121 **I:** ...more natural, more instinctive?

122 **A:** Yeah. Like this is my... you know, this is like a... metaphorical world, kind of. ... See, this [gestures to *Miscellaneous*] is like one continent. [Gestures to *Academic*] and this is another continent. [Nudges *Trash*:] And here's the landfill!

4.2.2 Subject B

Differences for Subject B. Remember that all of **B**'s Dynapad sessions preceded those of **A**, so Dynapad was less developed for **B**. The following list summarizes important differences for **B**:

DT Interface: Rather than the normal mouse-driven interface, **B** used the DiamondTouch table, described in 3.38.

More bugs: Because Dynapad (especially with DT interface) was still in development, many bugs and breakdowns consistently interrupted **B**'s sessions, especially her earliest ones. Additionally, several small interface details were found to be ineffective with the DT table, even when Dynapad operated as expected. Details which can be linked to an impact on behavior will be noted below.

Fewer features: As with most of **A**'s session, **B** had no labels. Also, the two-button carry-while-navigating (3.3.2) technique did not yet work with the DT buttons.

No initial organization: All of **B**'s documents were collected into the same directory before importing them into Dynapad. Therefore directory-brushing carried no information, and was turned off throughout **B**'s session.

Two phases: In addition to her initial collection of 155 documents, **B** added a second batch of 57 in session 3.

More images? Almost all of **B**'s documents were research papers or class materials (unlike A, with many personal documents). Therefore their portraits generally included more images, and were potentially more identifiable at small scales.

As with subject **A**, many of the critical phenomena in **B**'s activity can be seen in her first session. Look ahead to Figure 4.31 (page 208), and notice two things:

- The vast majority of items are still in the reservoir, which has been placed into a timeline tray (near the bottom of the workspace).
- The initial geography of the workspace has been established by the few items which have been extracted. But these are extremely diffuse, an order of magnitude farther apart than they need to be to insulate them from each other.

Let's trace the development of that state.

Session B1

B begins by putting a lens-tool over the entire reservoir grid whose default settings display a loosely-clumped timeline (see 3.6.3) of the documents' acquisition dates; Figure 4.25 shows the result.

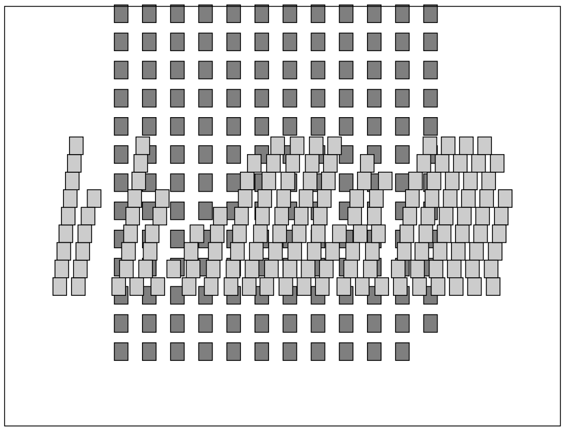


Figure 4.25: Timeline lens (transparent here) over initial import grid

Why **B** chooses to begin with a lens is unclear. But since **B**'s collection had no initial structure, leaving the grid items in effectively random order,[8] the organization produced by a region-tool's automation was all the more important as a scaffold for reflection.

Lens Trouble. A problem with the lens became immediately apparent. After looking at the arrangement for a few seconds, **B** tries to lasso and extract the leftmost grouping. Instead of starting a selector, the action moves the lens and rearranges the portraits within.

Remember that the portraits projected in a lens are proxies (see 3.6.4), "fake" copies of their sources behind the lens. The proxies can be zoomed and their portraits rearranged, but not moved independently; they are "sealed" in the lens. Therefore a lens has solid-like behavior (3.2.4) when dragged.

[8]The grid items were actually sorted by filename, but those were rarely chosen by **B** and had little consistency.

Even if lenses were changed to allow selection of the proxies (which would perhaps transfer to the corresponding sources underneath), the sources cannot be manipulated through the solid-like lens. The lens must be moved to access them, discarding its projected arrangement. Lenses are an example of a rare (in Dynapad) disconnect between visibility and accessibility; items can be seen but not touched.

Discovering this, **B** rejects the lens:

1 **B:** I... want to be able to interact with these things, like the clusters that form here <*the lens*>, pull those out... so I would use a ... magnet[9] for that, or a tray, or what would I use..."

2 **I:** Probably (a) tray...

As it turns out, **I**'s suggestion was probably bad advice; a stamp would have been more effective, as we'll see. But in response, **B** replaced the lens with a timeline tray, shown in Figure 4.26. Then **B** began extracting from this portable and automated reservoir.

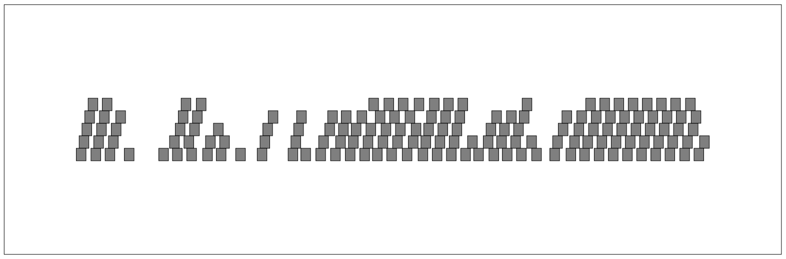


Figure 4.26: Lens replaced with timeline tray *(B1:03:46)*

Emergent Drift and Dispersal. **B** began at the left of the timeline, with the same group she had tried to circle in the lens. She began several piles with these items to the left of the tray (Figure 4.27).

Soon however, **B** zoomed in close to the right end of the timeline, and began "panning" leftward while remaining at a close view (the red box in Figure 4.27). In fact, she wasn't panning at all, but dragging the tray gradually to the right. Since panning uses a "ground-dragging" metaphor, both actions looked the same from **B** close perspective. Even if **B** had seen the edge of the tray, because the Dynapad surface has

[9]A stamp

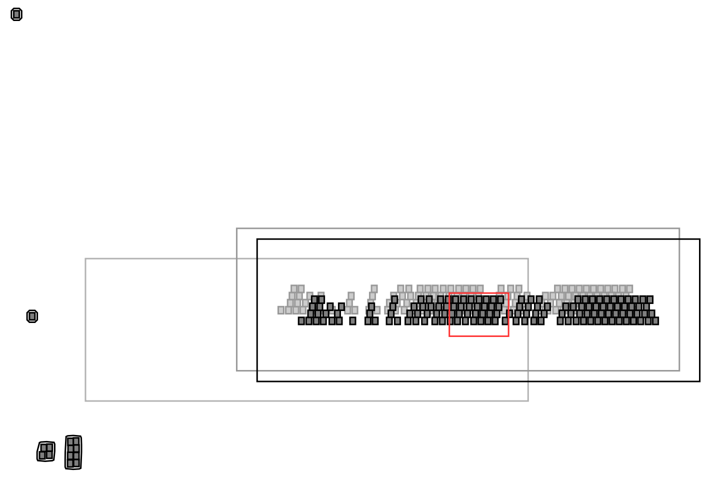


Figure 4.27: Drift caused by close-up moving instead of panning *(B1:17:54)*

no texture there would still be no visual cues that the tray was “drifting” through space, leaving the earlier extractions behind.

When **B** extracted the next several items, placing them just outside the tray’s left end, these were some distance away from the old extractions (Figure 4.28). Relative to the established landscape, these landed in completely arbitrary and unintended positions.

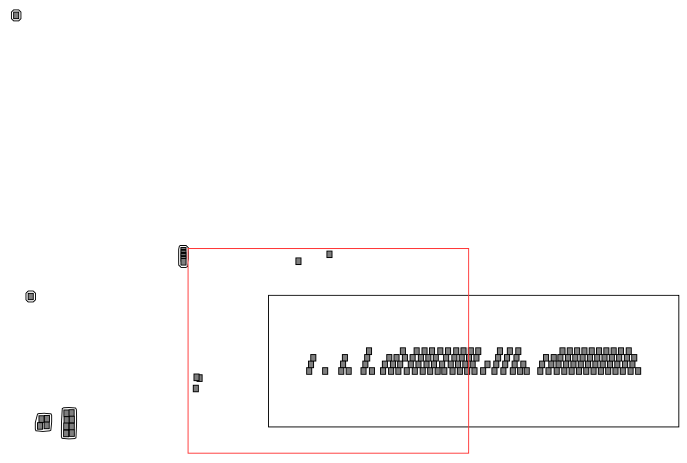


Figure 4.28: New extractions follow drift *(B1:22:06)*

Now **B** decided she had extracted enough for the moment, zoomed out to the far stop, and casually pushed the tray aside. But the earlier drift, combined with the tray’s length, gave the set of all objects a broad footprint, and the wide-zoomed view was wide indeed, with additional empty space at its margin. This allowed **B**'s careless dismissal of the tray to send it a considerable distance (Figure 4.29) — leaving the workspace with an even bigger footprint.

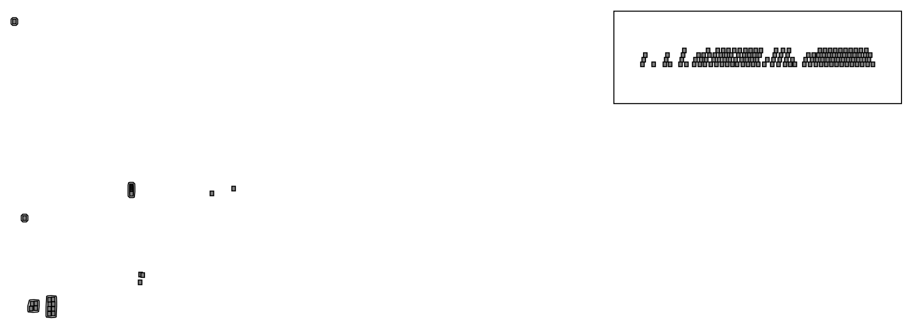


Figure 4.29: Incautious dismissal of reservoir as attention shifts

After consolidating several of her extractions, **B** again zoomed out until the tray was visible, then pulled it toward herself, to the front edge of the table, only then zooming closer to it. That move too displaced the tray a large distance (relative to the extractions). It ended up not significantly farther away than before, but its placement was again *unplanned*, the result of a sudden, pragmatic move much larger than any of the deliberate movements of documents.

B then began to unload more documents at this new, haphazardly-established location (Figure 4.30). As a result, the developing organization's geography is dominated by relatively thoughtless actions, while the meaningful adjustments are insignificantly small.

Indexicals in an unstable geography. Another casualty of this volatile geography is the last extraction, at the upper left of the tray in Figure 4.30. **B** has determined that it belongs with "Goodwin", a paper now far up to the left. It's too far and too much work to find it now, **B** decides, so instead she places it as a *pointer* from the tray toward its eventual destination. But once the tray is moved again, that information is lost; its now-isolated position becomes meaningless.

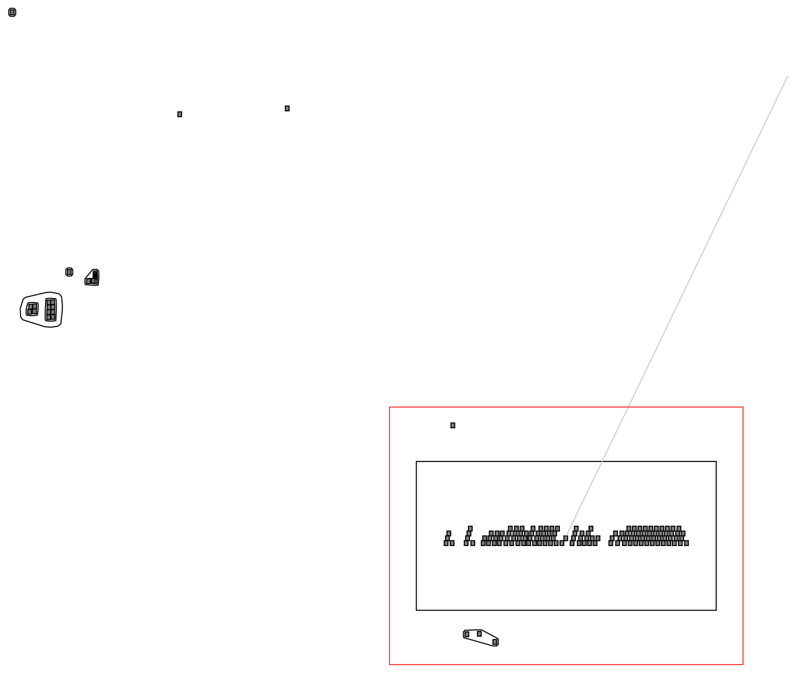


Figure 4.30: Pragmatic move, drawing reservoir closer *(B1:24:50)*

Scattering around the Tray. Consider also the trio of documents in a clump just below the tray. Why were they placed below? Since the tray was centered in the view, with equal free space above and below, I infer the reason to be that on the DT table, it was easier to reach below than above.

Furthermore, the distance between the singleton above and the trio below is a consequence of the tray's height, which at this point is even wider than necessary from an earlier unintentional resize. Again, when the tray eventually moves, this wide separation is completely disconnected from its motivation.

In short, **B** experiences a disconnect between *effort* and *impact*: careless moves have huge consequences for the geography, while numerous deliberate moves have little impact. The dominant spatial relationships are largely noise.

The Consequences of Diffuse Space. Ironically, although one of Dynapad's strengths is to afford unlimited space, providing *too much* space can be detrimental. A widely dispersed workspace incurs the following costs:

- The collection's effective visibility is lowered. Either the items must be very small when all are in view, or more navigation effort must be expended to see them at closer views.

With empty space between objects, valuable pixels are wasted; there is less information per pixel. The only solutions are

1. to "wrinkle" the display: distort the workspace to suppress empty space and allocate unused pixels to objects (which creates other complications);
2. to help the user better manage space, preventing excess empty space from creeping in at all.

- As with visibility, effective *access* is lowered. It requires (slightly) greater mechanical effort to move items, both because travels distances are greater, and because the chance of manipulation errors (e.g. missing) increases when items are small. This cost is even greater for **B**, using the DT table, whose pointer precision is already low.
- The contiguity of higher-order groupings is lower. Perceptually, this decays aggregation; visual gestalts and patterns become harder to see. Mechanically, this increases the cost of selecting sets (via lasso, for example).

The Causes of Diffuse Space. Finally, after several more extractions, **B** concludes session B1, with widely diffuse state shown in Figure 4.31. Now that we've seen the history behind this arrangement, let's review the aspects of Dynapad which led to such wide dispersion:

Blank surface A Dynapad workspace lacks any absolute reference frame; its surface has no texture, edges, or visible center. The only landmarks are the documents themselves.

Mobile reservoir For **A**, the elements of the reservoir provide a stable reference geography, and as that feature is depleted, it's gradually replaced with a landscape of processed items. But **B**'s decision to place the reservoir in a portable tray eliminated the only source of stability before a critical mass of extractions could provide an alternative reference framework.

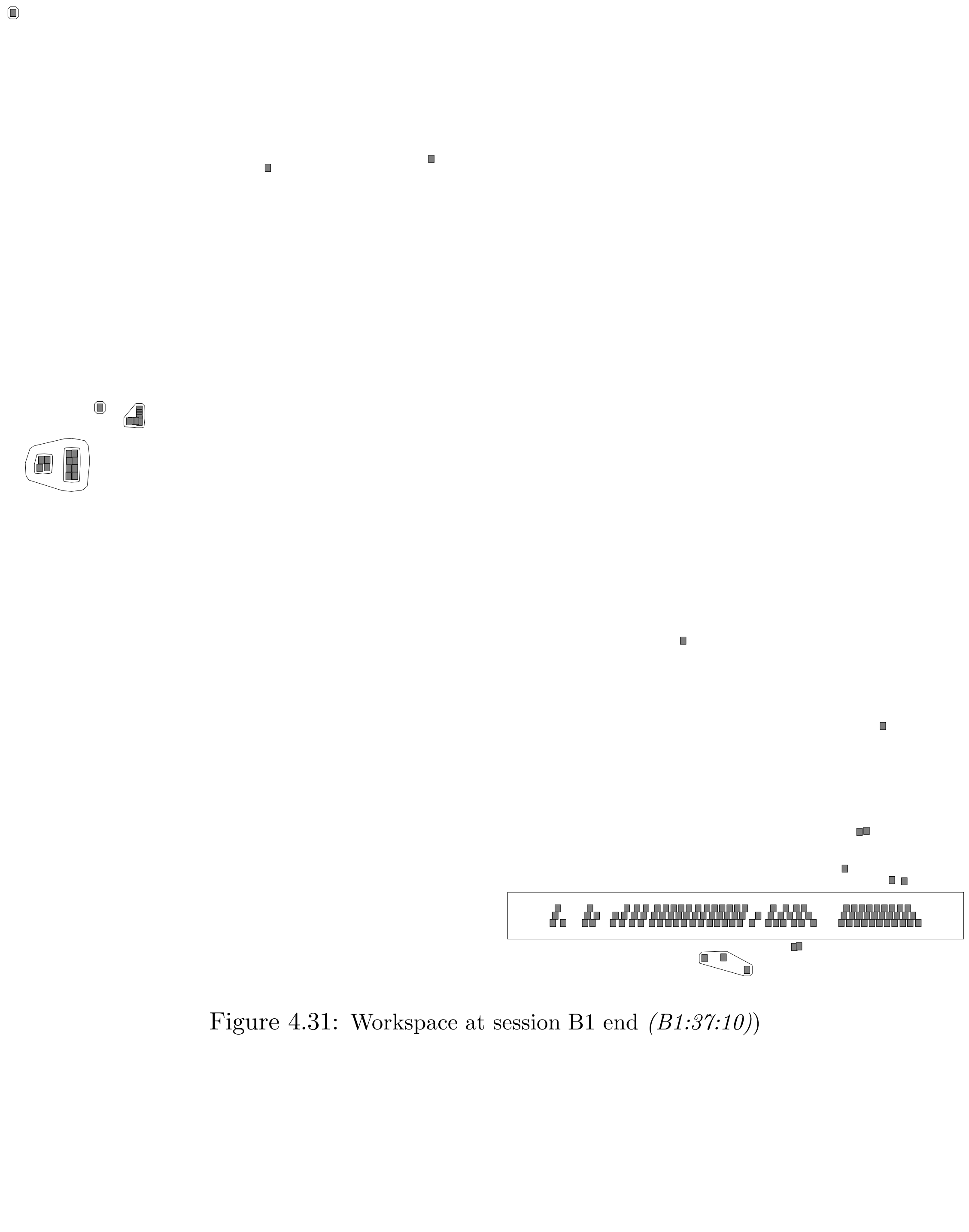


Figure 4.31: Workspace at session B1 end *(B1:37:10)*)

Inflated demand for tray **B**'s decision to use a tray arose in part from a need for its arrangement effect, not its syntactic aspects as a container (cohesion and portability). But the desired arrangement effect is always combined with the syntactic idiosyncrasies of some tool, and at least one of the alternates (the lens) proved equally obstructive.

Overloaded action vocabulary As a user shifts attention, she will naturally act to suppress certain items and emphasize others. But when the action space includes only repositioning, such attention-directing and pragmatic moves become conflated with meaningful (information-encoding) moves.

Unmoderated expansion As we saw earlier with **A**, Dynapad fails to regulate the consumption of empty space. Zooming out to the far stop leaves increasingly more room at the view margins, letting the workspace expand at an accelerating rate.

Artificial gravity On a horizontal surface, such as the DT table, the need to reach physically to locations creates a "downward" bias in the workspace. Other considerations being equal, a user will tend to pull objects closer, creating an artificial "gravity" which may widen dispersal.

Object drift At close views, dragging and panning are indistinguishable. In a homogeneous and unconstrained space, mistaken dragging will cause objects to drift randomly. And such dragging is more likely with an interface in which panning requires more work (as with the DT table, where navigation requires a second hand pushing an often-unresponsive button).

Inflated footprints Finally, a tray in particular causes extra dispersion since objects must be moved clear of its edge when extracted. Since its size is unregulated, its edges may be unnecessarily far apart.

More Tray Troubles. Besides its contribution to dispersion, a tray creates an additional burden: it does not merely offer a layout but enforces it, refreshing after each interaction and forcing its members back to their computed positions. This has several unfortunate consequences:

Delay Each possible move of an object in the tray, including all the possible errors (see 4.2) cause significant delay as the tray refreshes (especially slow because it holds the entire reservoir). This delay is self-reinforcing: as behavior becomes unresponsive, **B** tries additional prodding which triggers additional refreshing.

In response, to reduce errors of mis-targeting small objects, **B** tried zoomed closer to the tray. But then often its edge was out of view, and the items could not be extracted in a single move; the DT interface could not pan and carry, but dropping a target within the tray would force it back into the layout. Eventually **B** compromised by resizing the tray to be as narrow as possible.

De-centering Because of dragging and panning were indistinguishable at close scale, **B** would often re-center an object in view by dragging it (instead of panning) slightly. When zoomed into a tray, it would evaporate (see 3.2.4), leaving no cue that the target object was within a special space. Then, whenever **B** tried re-center it, the tray would send it back to its designated place, off-center.

Disorientation For **A**, drawing from the inert reservoir left a hole, and the grid soon acquired an internal texture as a reference frame.

But for **B**'s reservoir, the tray's automatic layout would attempt to close holes when items were removed, not only destroying that landmark but moving other items as well, sometimes great distances if they wrapped to a new row or column.

In one extreme case, **B** looked at two adjacent documents, intending to extract both. Since *shift* was inconvenient (requiring the keyboard), **B** didn't try to select and move both together, instead moving one at a time. After dropping the first, she moved her finger back to the second — which was now gone, moved elsewhere during the refresh.

Differential Consequences of Region Tools. Ironically, the most effective region-tool to use would probably have been the seemingly scary, destructive one: the stamp. "Stamping" her initial grid into a timeline would have offered the same automated structure as the lens and tray. But unlike the lens, the stamp (once removed) allows easy accessibility to the grouped items. And unlike the tray, the stamped arrangement would have been stable (not rearranging during interaction), anchored (thus preventing drift), and free of boundary-clearing hassles.

But of course, this strategy would have its own disadvantages. Without a boundary, the reservoir might need other source of insulation. Visual coherence might be enough, as with **A**'s layout grid, but **B**'s timeline has a weaker visual gestalt for two reasons. First, the timeline layout can appear messy, especially when crowded. Second, once items are extracted from the now-static arrangement, their holes further obscure any regularity which distinguish the timeline.

This lack of insulation could be solved by simply drawing a visual line around the static reservoir. Such drawing features are available in Dynapad but were disabled in these sessions, their value unexpected. Alternatively, a clump would work as well.

In the next session, **B** discovers this trick, using a stamp to arrange items, then wrapping them in a clump for insulation.

Session B2

B's first effort in session 2, except for a few moves, was to consolidate the diffuse workspace, bringing the outliers closer to the reservoir tray. The most dominant elements at this point are the three *courses* piles, which **B** moves to the right end of the tray. These become the primary landmark around which the rest of the organization eventually follows.

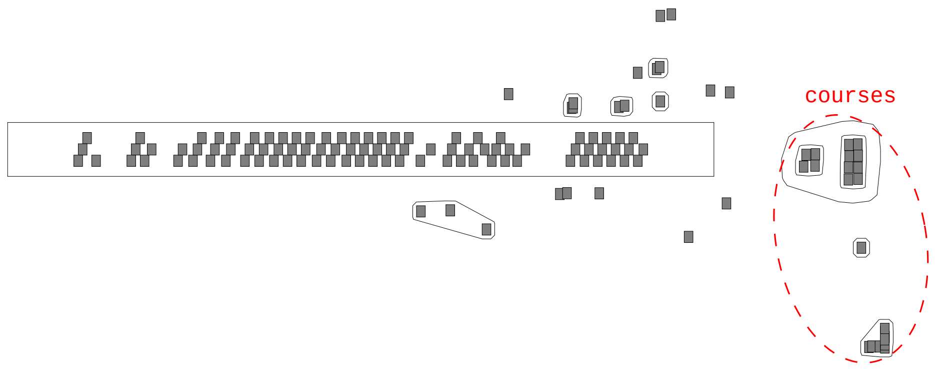


Figure 4.32: Main workspace area, surrounding the reservoir. Three *courses* piles have been brought close with their earlier linear configuration rotated.

Changing strategies. Increasingly aware of the obstacles in using a tray for the reservoir (see 4.2.2), **B** soon adopts a different strategy. She move the reservoir away to the periphery, below the main workspace, and removes its tray, as in Figure 4.33.

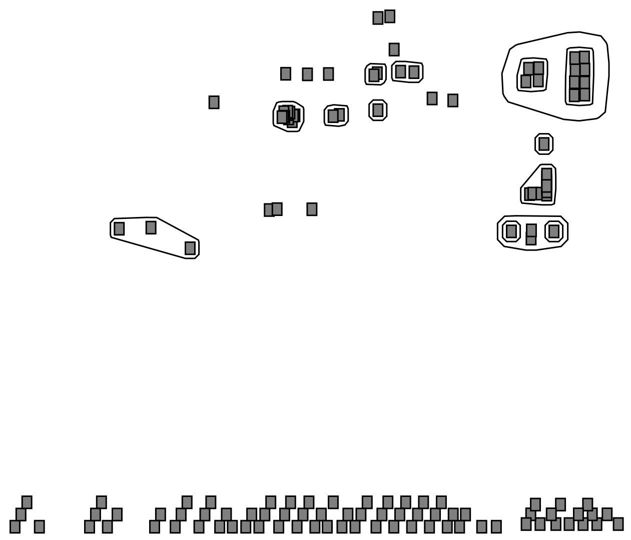


Figure 4.33: Reservoir removed to foreground, its tray dismissed. Only the main workspace area is included here.

Since the residual timeline (left over from the tray) divides the reservoir into several distinct "eras", **B** decides to focus on one group at a time, starting with the most recent. She selects the rightmost group and brings it back to the central area (Figure 4.34). **B** then says of these items:

3 **B:** Now I just want to make them... kind of stay, so maybe I should... pile them [wraps them in a pile].

It's not clear what **B** intended. They would stay in place even if they remained loose, and **B** seems to have enough experience to understand this. So she may be alluding to the pile's ability to *cohere* them together, protecting the arrangement against accidental disruption. Though not directly reflected in her comment, this pile has the added benefit of *insulating* the unprocessed members from the surrounding visual complexity of the earlier extractions.

Having the chronological organization from earlier but without the disorienting refresh, **B** proceeds much more fluently than before (and admits, "Yeah, you're right,

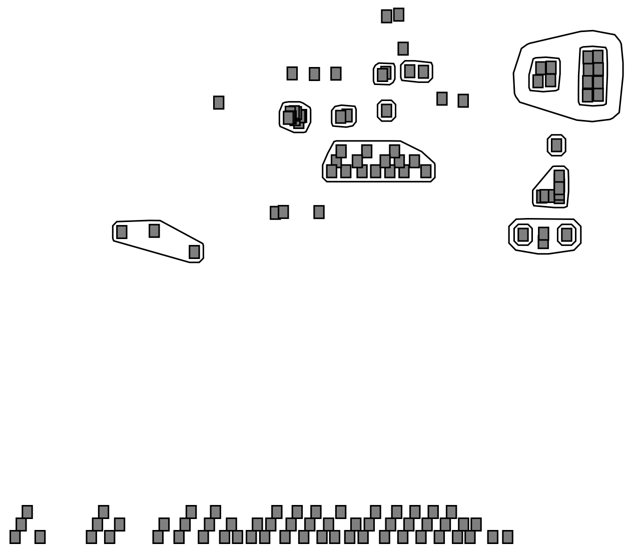


Figure 4.34: The first sub-batch is returned to center for processing *(B2:34:52)*

this is kind of easier”). **B** distributes the local batch into the surrounding categories and eventually wraps the group of courses into a common *courses* pile (Figure 4.35).[10]

Tidiness (scene 1). After making the courses pile, **B** muses:

4 **B:** I would like a real pile, you know, like a “file” pile.... Like a, you know, square pile. Like a square pile that has, you know [gestures around perimeter of *courses*] (a) nice square, and then [gestures in descending rows of *courses*] lines, and stuff like that. Like a (bookshelf?).

5 **I:** You’ve seen too many computers.

6 **B:** No, just because... this is starting to be neat and I want it to look neat.

7 **B:** I mean, I guess I could do that by... [begins to adjust documents in bottom row] arranging the ... [Struggles with fused piles.] I just want to make a square.

B follows this with a long series of careful “stretching” moves: moving and re-spacing the inner piles to have the same width all the way down *courses*, until she achieves the arrangement in Figure 4.36.

8 **B:** [After much fussing with the topmost, trapezoidal subpile:] That might be good enough... Maybe I just have to take a paper and copy it multiple times (in the line) to delineate the boundaries.

[10]In one case here, and others later, she demonstrates pile-spanning or “nearness” encoding.

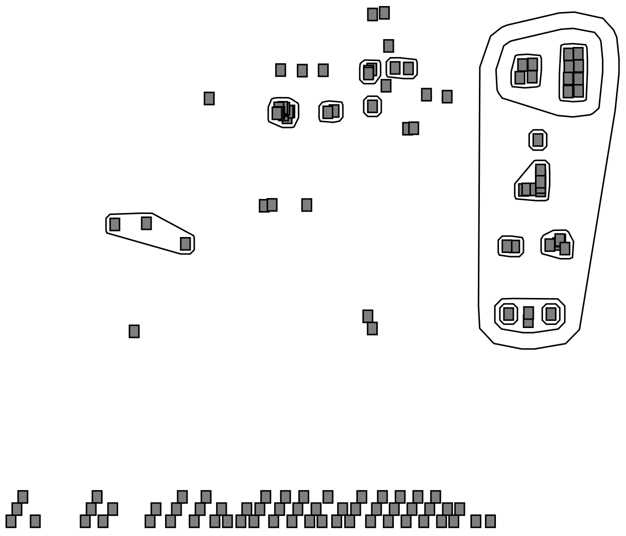


Figure 4.35: Distribution of sub-batch and wrapping of *courses* pile *(B2:43:15)*

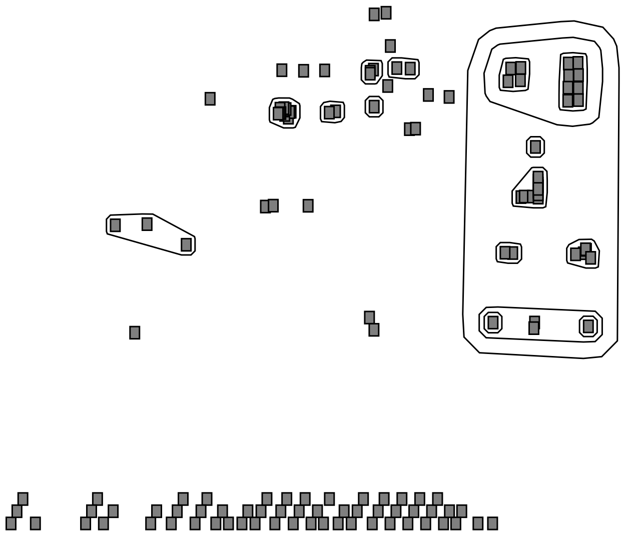


Figure 4.36: Tidying the *courses* pile *(B2:45:18)*

B never actually does this, although later she does something similar, using disposable copies of a portrait to shape a pile (see 4.2.2). But this idea is similar to the aggregate letters of **A**: both subjects make creative and opportunistic use of fine-grained manual arrangement to achieve an aggregate effect that improves clarity. In **B**'s case, the effect does not label piles but rather "polishes" them, both accentuating their structure and annotating them as "finished".

Using Temporal Context. After a few other adjustments, having exhausted her local reservoir, **B** brings another (large) batch from the main reservoir to the central area, protecting it with a pile as before (Figure 4.37).

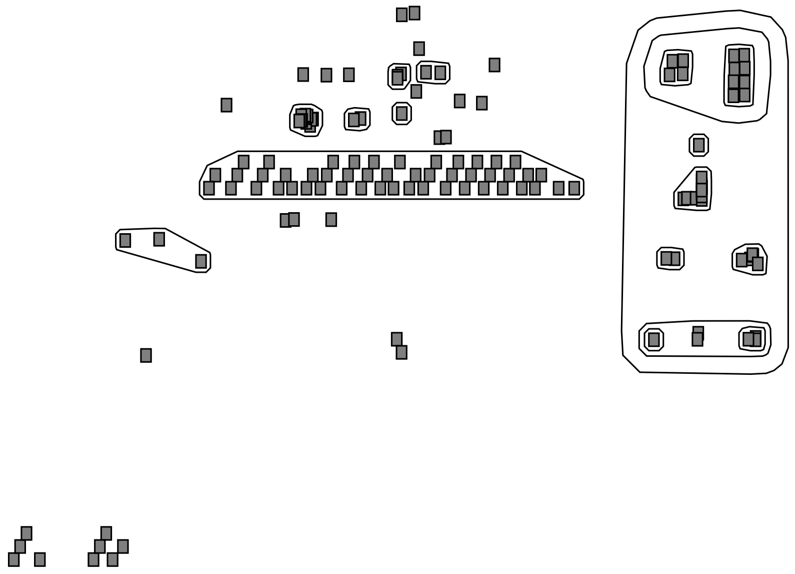


Figure 4.37: Re-centering a second batch *(B2:48:02)*

B wants to see the time labels for the batch. So she covers them with a stamp (actually putting it into the long pile with them), and the dates appear there. Having seen what she needs, she deletes both the arranger and the pile, then separates (downward) a contiguous block near the right end of the timeline which the labels showed to have come from a single day (Figure 4.38). Meanwhile, two documents from the far right end are left behind, and forgotten for a long time since they are no longer visually contiguous with the reservoir and look like processed items.

B characterizes the extracted block as follows:

9 **B:** The reason I have these is that I thought it would be interesting to just browse... things like *Journal of Cognitive Neuroscience* and *Journal of Cognitive Science* to see what kind of stuff there was in there. So I just went ahead one day and collected a bunch of papers from those two journals... all from one issue, and started looking through them. I never really looked at all of them.

10 **B:** So I don't really know where to put that stuff, because... it's like... kind of a game... it's not courses and it's not research, I don't know... [picks it up, looking for a destination] weird [puts in far NW corner].

This example demonstrates clearly the value of the automatic timeline arrangement. **B** was able to process an entire group of 16 documents at once, many of which she had never even seen. Had they remained in the original import grid, their filenames would most likely have scattered them widely, and they would not have been considered as a unit. But appearing here as an ensemble, with visible clues to their synchrony and temporal context, made them easily recognizable. Furthermore, the timeline had already enriched their relative configuration to be exposed, contiguous, and serial.

Due to a bug, this final move to the northwest was not logged correctly and Dynapad crashed here, so session B2 effectively ends as depicted in in Figure 4.38.

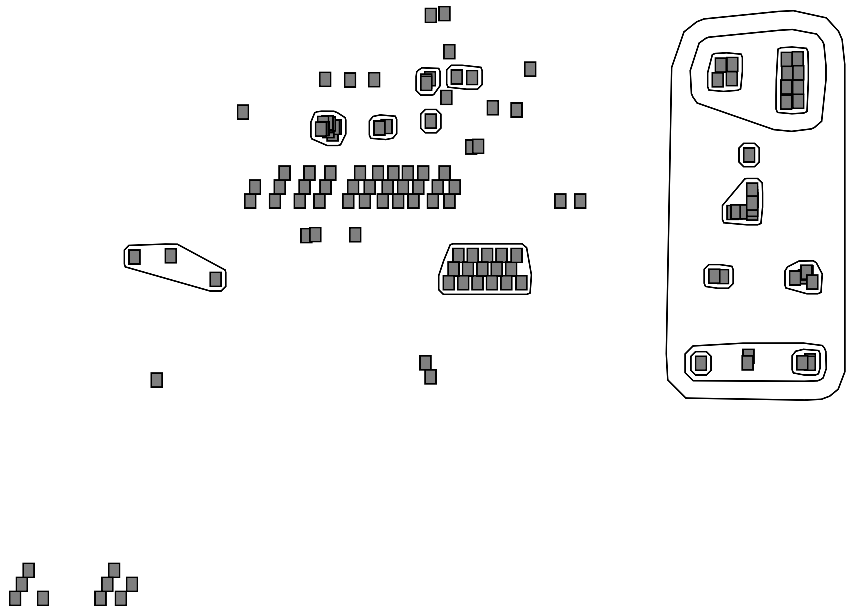


Figure 4.38: Using temporal context to split the second batch; end of session B2

Session 3

Interestingly, at the start of session B3, **B** did exactly the same thing as at the end of B2, 13 days earlier, moving the *journals* block to the northwest corner. In B3, **B** made no reference to the earlier episode, and when **I** pointed it out, **B** expressed surprise. It could be that **B** was re-enacting an implicit memory, or that something about the workspace suggested it, but I have no evidence of either.

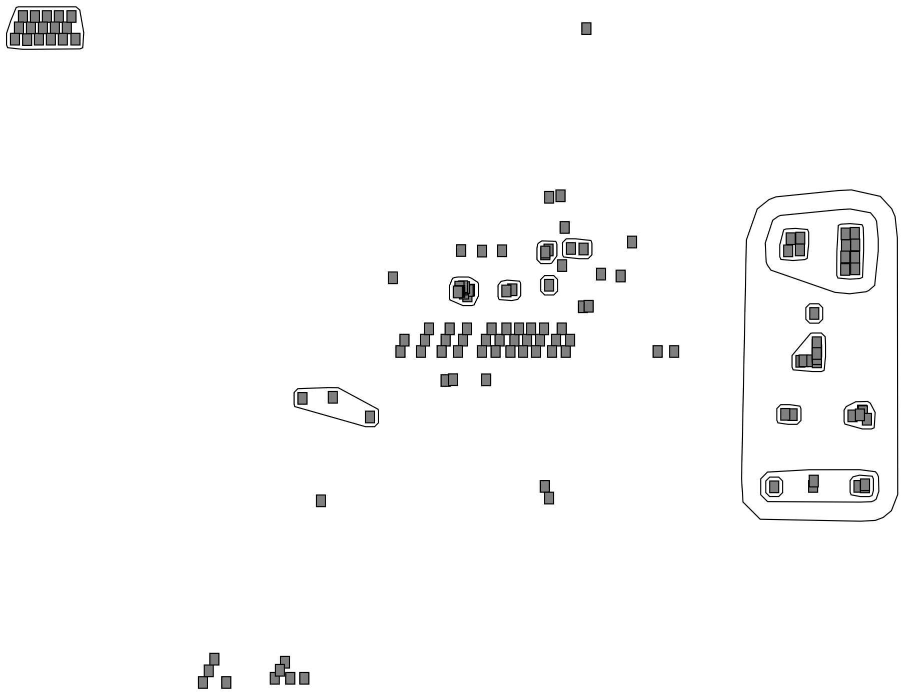


Figure 4.39: Starting session B3: The *journals* block dragged away to NW

The Value of Portraits. Next, **B** begins a process which eventually divides the remaining items of the local unexamined batch into two categories of research documents:

11 **B:** You know what? I think I'll need to do something different because this takes so long... I think I'll just find all the ones that have something to do with ERPs and EEG, and try to... work from there.

At this point, **B** extracts the ERP-related portraits introduced earlier in Figure 3.2. She initially gathers them into a line below the main batch, and then wraps them in a small timeline tray (Figure 4.40). Then she moves the tray farther below the batch ("I'll look at those later"), and in its place begins a new pile for *my research* (Figure 4.41).

Why did **B** make a tray for *ERPs* but a clump for *my research*? She says nothing to suggest the motivation in either case, but one factor may be that she already has a few *my research* documents wrapped in piles near the batch, waiting for a category to unite them. Since they are already subdivided, and piles cannot be placed into trays, **B** maintains the subdivision within an outer pile.

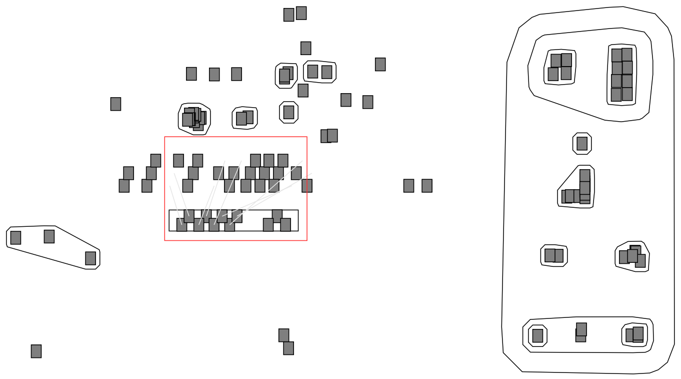


Figure 4.40: Part of batch moved to a small timeline tray. The chosen items are the sample portraits of Figure 3.2.

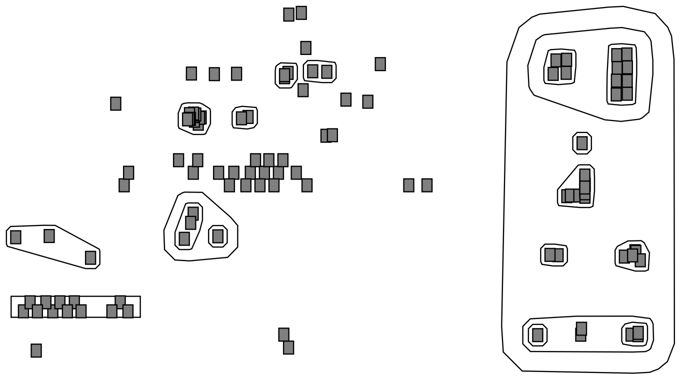


Figure 4.41: Small tray placed aside; local pile begun with new topic (*my research*?)

Thinking ahead, **B** begins to stretch the new pile by incrementally nudging its two component piles farther apart:

12 **A:** [while stretching pile:] Now I can create a space for my research stuff.

After she judges it to be big enough, she moves it above the work area to begin adding *my research* documents (Figure 4.42).

After finding several documents which **B** decides belong to that pile, she redefines its definition:

13 **B:** Actually I don't think this is my research pile; this is my disfluency pile.

B continued to search through the remaining loose items, distributing them among the *disfluency*, *ERP*, and *courses* piles, enlarging the *ERP* tray when it became too crowded (Figure 4.43). When the loose items had been classified, **B** joined the piles

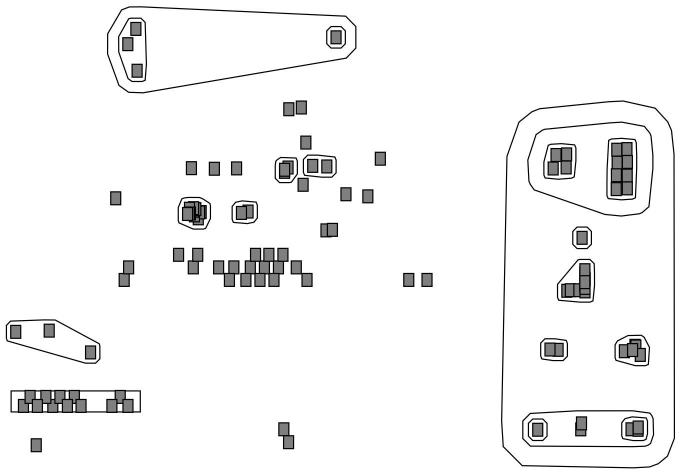


Figure 4.42: New pile moved and stretched to receive additional items. (Topic: *disfluency*.)

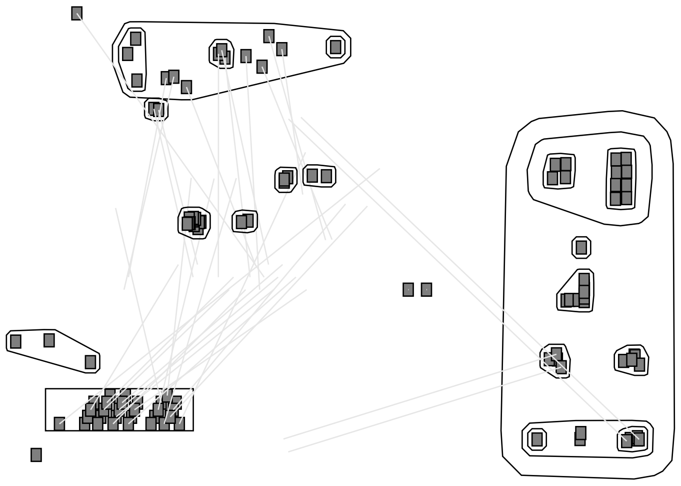


Figure 4.43: Unclassified items placed into stretched *disfluency* pile and *ERP* tray *(B3:17:58)*

into a larger *research* pile (Figure 4.44) and later moved it adjacent to the *courses* pile (Figure 4.45), joining it to the backbone of the organization.

Meanwhile, **B** iterated through the remaining items in the original reservoir, at the bottom edge of the workspace, and added them (mostly) to *courses* (Figure 4.45).

Finally, **B** gathered the leftover items, already wrapped in individual piles, and merged them with the main piles (Figure 4.46). At this point, **B**'s original collection was fully classified, although a second pass of organization would be needed in the two main *research* piles.

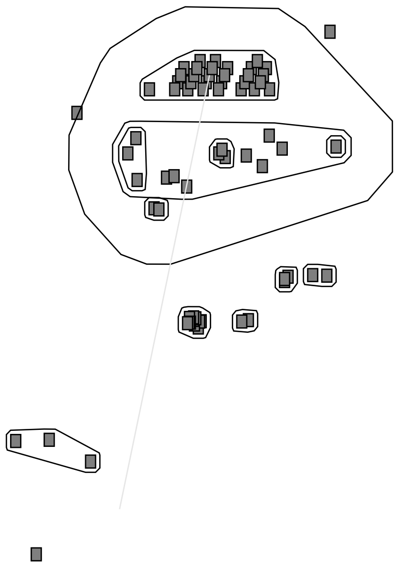


Figure 4.44: *ERP* timeline joined with *disfluency* pile into larger *research* pile *(B3:18:50)*

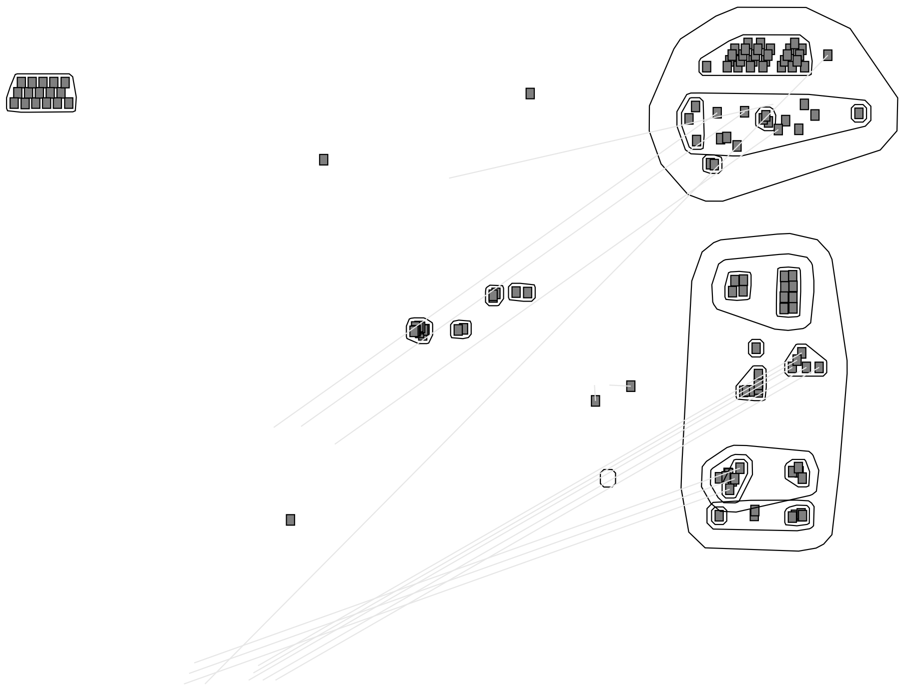


Figure 4.45: Return to full workspace view. A few central items and remaining reservoir distributed into *research* and *courses*

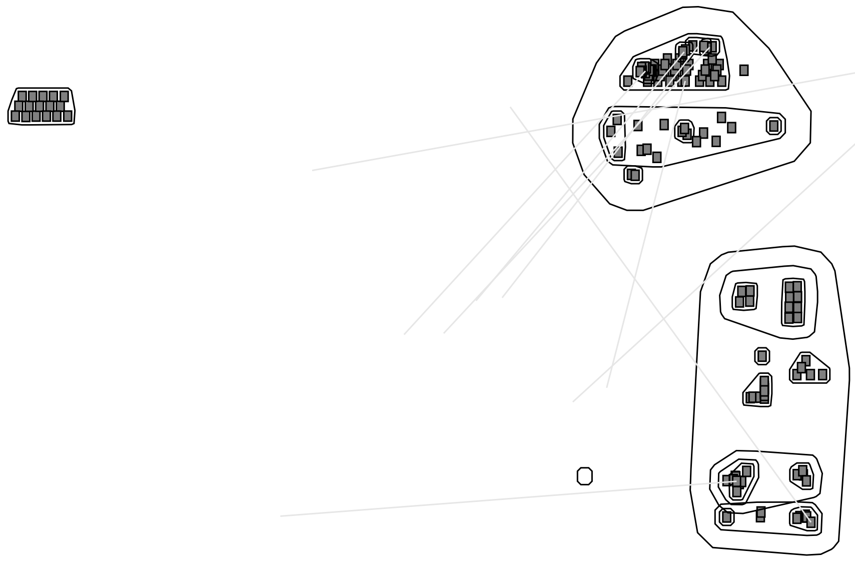


Figure 4.46: **B**'s original collection fully classified *(B3:31:39)*

A Second Wave. Since her first session three weeks earlier, **B** had been collecting additional documents. With her initial collection processed, she now imported that second wave of documents. Having learned from her earlier experience with timeline tools, **B** immediately used an arranger-stamp to force the new batch into a timeline in the newly cleared workspace center (Figure 4.47).

Working from that reservoir, **B** then processed these documents very efficiently, examining them one at a time and placing them directly into one of the existing categories, with little intermediate staging. Her efficiency is probably due to a combination of factors:

- She had collected all these documents very recently and could easily recognize each;
- Her existing categories seem to be adequate, saving both the mechanical and cognitive effort of establishing new ones;
- The new reservoir was centrally located, with no distractors nearby.

B spent the rest of session 3 distributing the new items, ending with the state in Figure 4.48.

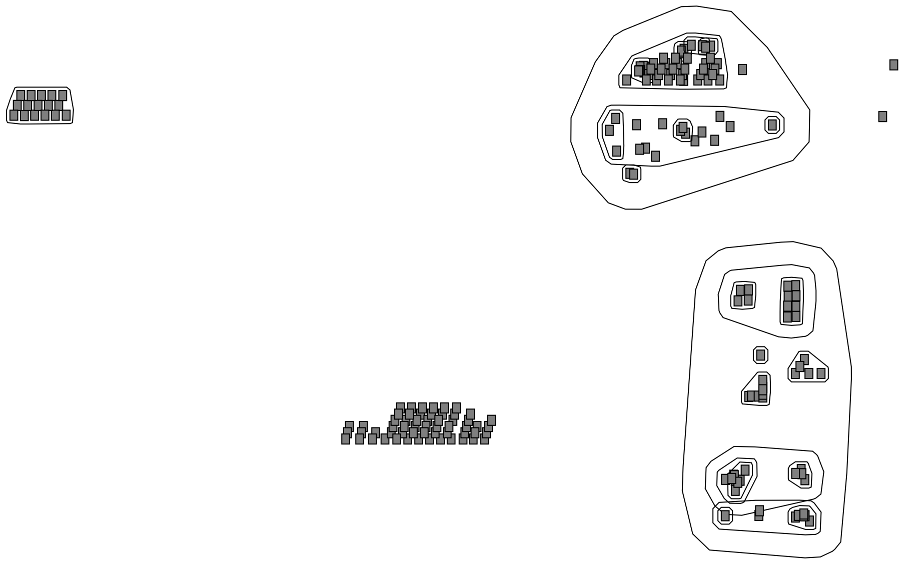


Figure 4.47: New batch after timeline arrangement by stamp *(B3:35:47)*

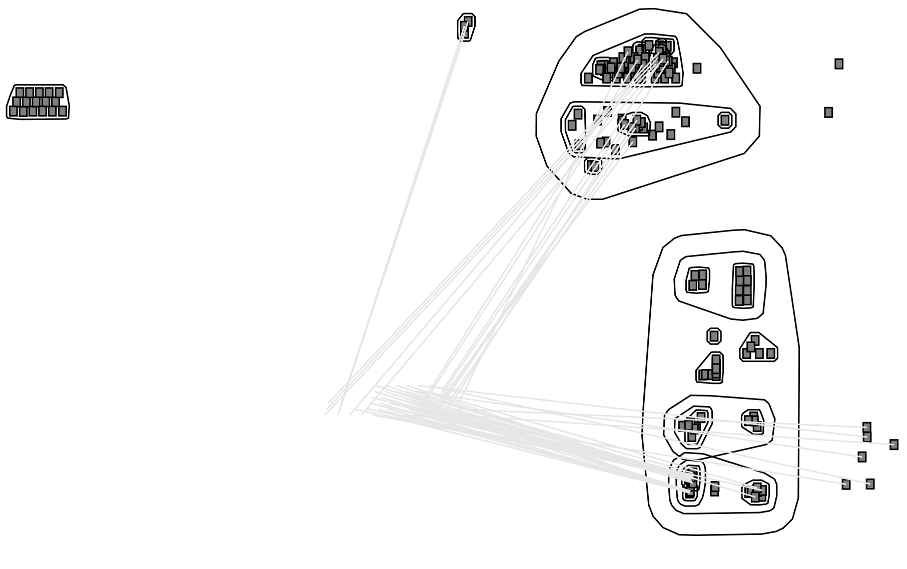


Figure 4.48: Workspace at session B3 end *(B3:52:35)*

Session 4

In session 4, **B** begins to clean up her *research* pile. To begin, **B** extracts the *disfluency* subpile to the center foreground; a close-up of that area is shown in Figures 4.49–4.51.

More Creative Aggregation. **B**'s first action with the pile is to extract one item, make four copies of it, and put a copy in each corner of the pile, effectively "pinning" it in place as a rectangle, shown in Figure 4.49.

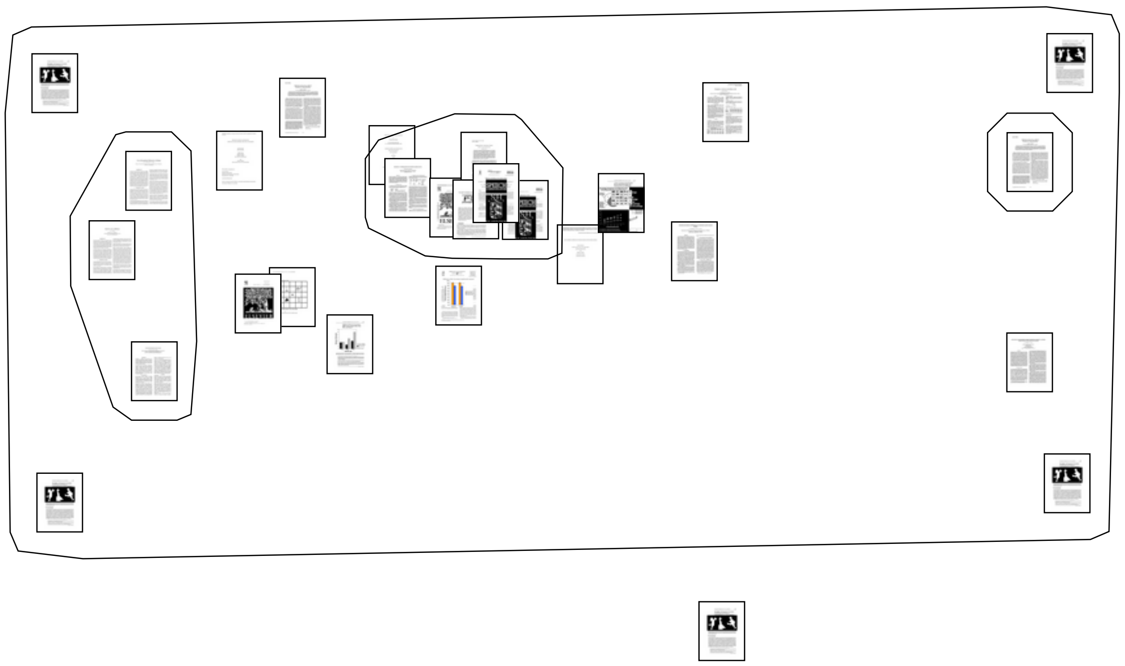


Figure 4.49: Pile anchored at corners using redundant copies of a representative item *(B4:02:50)*

14 **I:** This is an unusual strategy... Is this for aesthetic considerations only?

15 **B:** Oh yeah... No, actually, the reason that I chose this one [points to original] is (because) it has the (tangrams figure) on it. And (that's) a very obvious indicator for the kind of (research) that I'm interested in. [adjusts slightly] OK.

16 **I:** So why do you want this nice ... square canvas?

17 **B:** Because I'm going to tidy it up.

B doesn't actually use the empty space in the squared pile, but treats it like a mini-reservoir and extracts from it, making subcategories in the area below (see Figure

4.50).

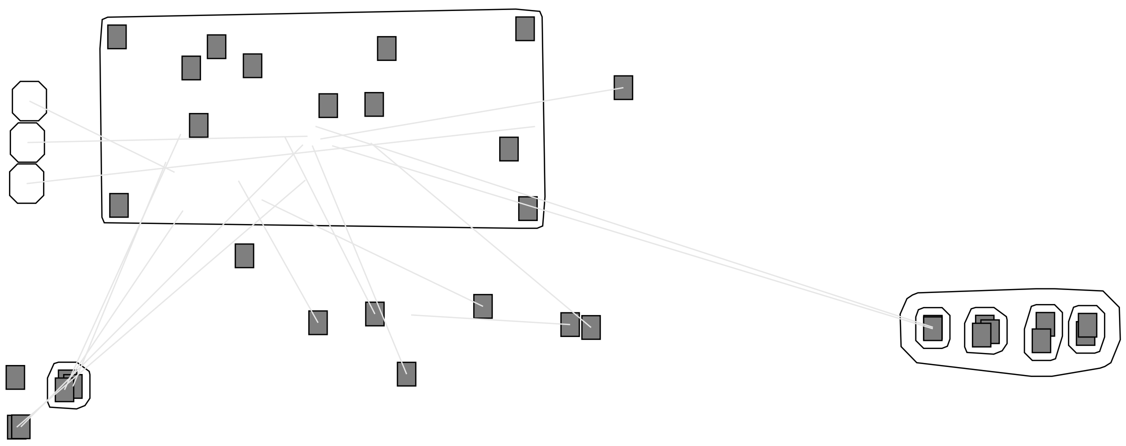


Figure 4.50: Same pile, still anchored, partially emptied. Note group of duplicates to right, and saved empty piles at left.

Figure 4.50 shows also two other interesting structures. On the left are three empty piles, formerly wrapping items within the now-rectangular *disfluency* pile, but no longer needed. Instead of deleting them, **B** has “stockpiled” them for later use. It’s not clear that such recycling saves **B** any effort, but their presences inspires a novel use we’ll see later in Figure 4.53.

Multi-Filing. On the right of Figure 4.50 is a special *author* pile, containing four (eventually five) subpiles of duplicates of items extracted from *disfluency*. **B** organizes the bulk of items by topic, but sorts these duplicates alphabetically by author, her first use of multi-filing (see 3.4.2).

Interestingly, while **B** abuts documents in the topical piles to leave them visible, she stacks the duplicates in the *author* pile, leaving only the topmost visible. **B** offers no explanation for this difference, but she does describe her conception of these duplicates:

18 **B:** I guess what’s different about these papers from some of the others is that I actually associate... theoretical stuff with these people. So, like... there’s ideas I would associate with them and not only facts.

So to speculate: **B** may have a clearer identity in mind for these *author* piles than the topical piles. The purpose of leaving the topical piles exposed may be so that the visible members can remind **B** of the scope of the less-defined topic, with each member contributing to the group's *de facto* definition. But the *author* piles need only a single exemplar to communicate their membership.

Rectangular Tidiness, part 2. So what is **B**'s true purpose in squaring-off of her *disfluency* pile?

Immediately after stacking the last duplicate in the *author* pile, **B** quickly pulls all remaining documents from the square *disfluency* pile and replaces them with the newly-tidied *author* pile. At this point, the square pile is no longer a reservoir, but seems to be instead a "showcase" of finished work. All remaining work of sorting by topic has now been moved below that pile, where **B** gradually assembles the "descending columns" arrangement of Figure 4.51

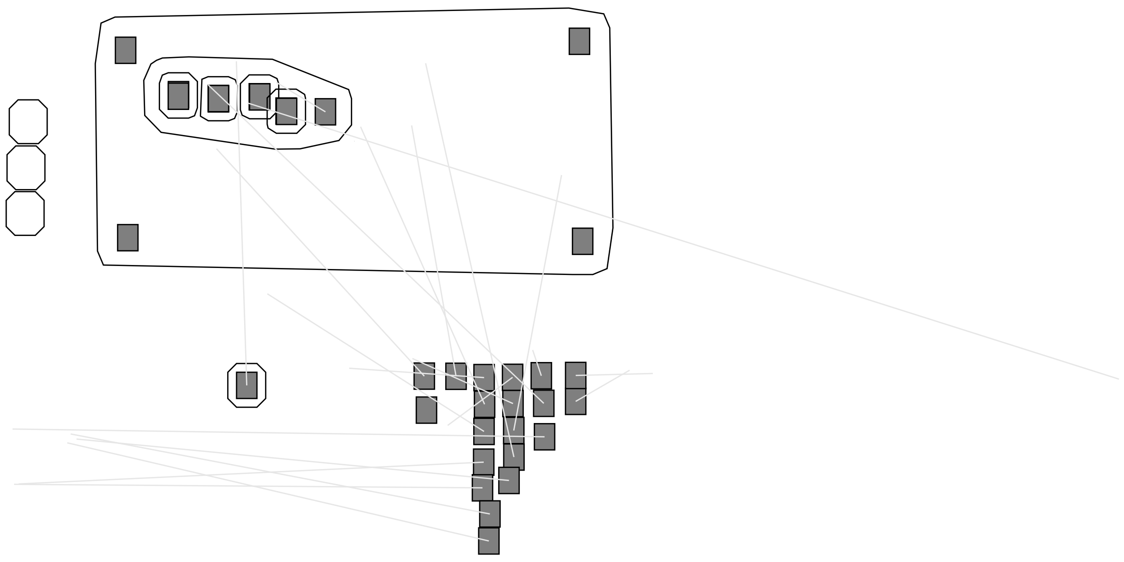


Figure 4.51: Same pile, still anchored, partially emptied. *Author* pile has been moved back to "neat" pile.

Note that **B** didn't use the physics of the squared pile at all; she needed neither to move it nor to stretch it, having plenty of space to work in. **B** only utilized its visual *demarcation*, a perceptual cue about the finished status of its contents. For this purpose, an inert line drawn on the workspace would work just as well. But because the interface offered no such feature,[11] **B** went to extra work adapting a pile to the purpose.

[11] Actually Dynapad does include a suite of drawing tools, but they are normally disabled in collection-management mode.

General Visibility. Earlier in the clearing of *disfluency*:

19 **B:** ...This is hard...

20 **I:** How come?

21 **B:** I think one problem I'm having is that... at this resolution where I can see all of them, I can't tell them apart; I can't tell what each one is. Like for this one [points to corner-anchor duplicate], OK I can, because I know the picture, but the other ones don't have characteristic pictures, so I just don't know what I have. So it's really hard to create groups because... I just can't see.

Effective visibility is reduced by the portraits' lack of images. And this set, like **B**'s early workspace, has low density. More space between documents means each is smaller when all are in view. Ironically, **B** has made the problem worse by nailing down the pile's corners; it occupies more space than needed, and its natural zoom size leaves its contents smaller.

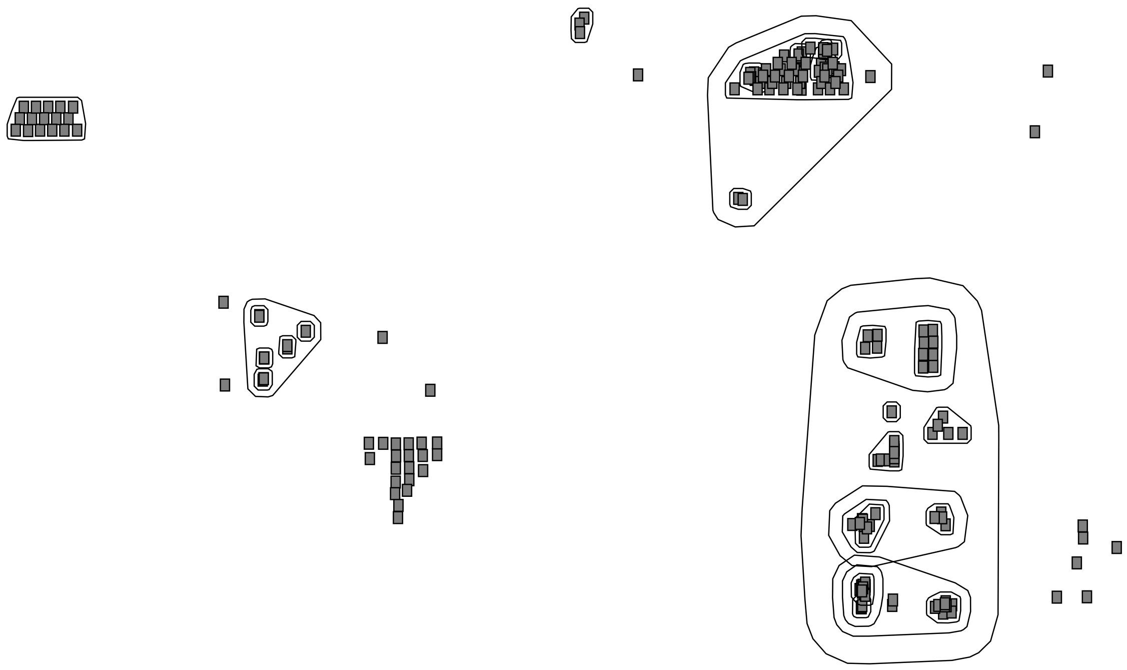


Figure 4.52: Just after deleting the rectangular pile; only its anchors remain.

Rectangular Tidiness, part 3. As a final touch, **B** wrapped the "descending columns" piles in a new pile. But still seeking tidiness, she decided to use the leftover empty piles (Figure 4.51) to square that pile as well, making it the same height as the adjacent *author* pile. Again **B** has made novel and opportunistic use of available resources to achieve a high-level effect (i.e. "polish") that Dynapad did not anticipate a demand for.

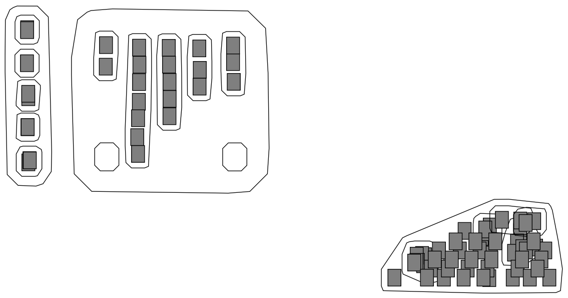


Figure 4.53: Close-up of "neat" piles contrasted with "messy" pile

She comments:

22 **B:** I guess, just,the reason... I mean, now I know this <*squarish pile*> is organized. Know what I mean?

23 **I:** Now you know that it's organized?

24 **B:** Yeah, because it looks neat.

25 **I:** OK.

26 **B:** [pulls rough *ERP* pile alongside] I mean, just compare this.

27 **I:** They do look different.

28 **B:** Definitely. I mean, here <*squarish*> I know there's structure there, and there <*messy*> I don't know.

The utility of manually-enriched microstructure is not merely for pragmatic reasons (to expose, serialize, and consolidate items). Clearly, **B** harnessed an *aesthetic* function as well, which served as an aggregate signal marking a region as "finished".

There are two ironies here. First, the contrasting "messy" pile was originally the product of automation (see Figure 4.43) but arranged in such a way as to appear untidy (then further contaminated by hasty manual additions). Second, the very act of automating a region potentially destroys its "finish": unless its contents are scrutinized by hand, item by item, the user may not have confidence that they're truly processed.

Surely a region-tool could support a benign "neatening" operation. Or more abstractly, how might it otherwise annotate a zone as "approved"?

This suggests a dissociation between "neatness" (visual regularity) and "solidity" or "confidence" (a history of engagement).

Kaboom! The End. Figure 4.54 shows **B**'s workspace late in session B4. After this, she went on to clean up the *ERP* pile. Unfortunately, Dynapad crashed during this process, something went awry in the log files, and the state shown here is the last (reliable) one available.

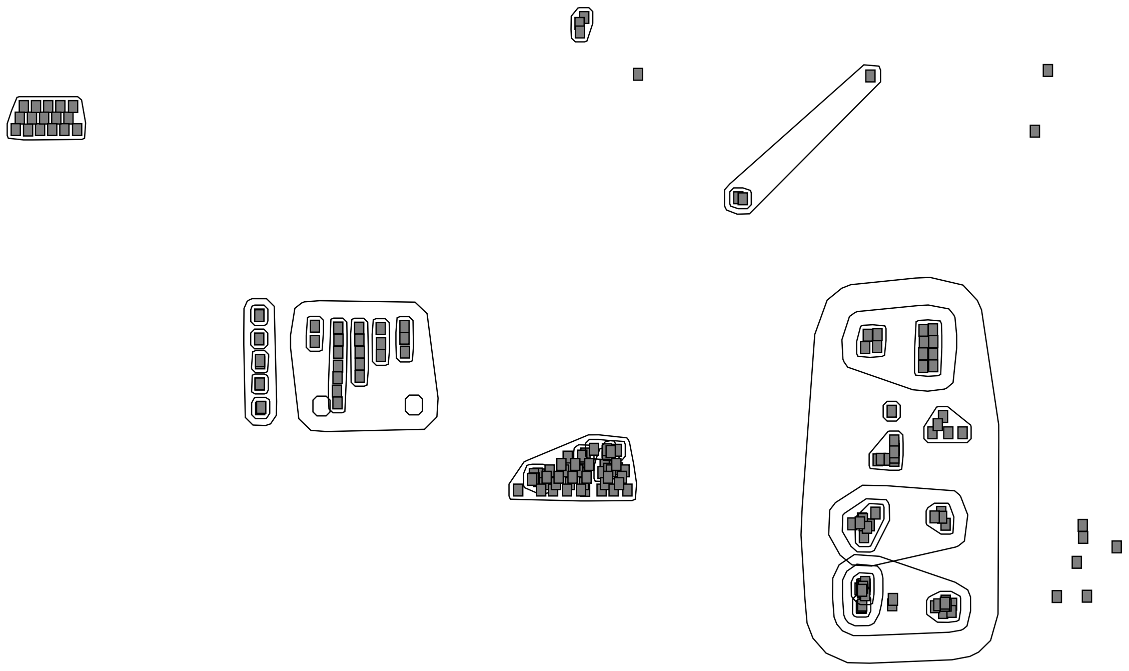


Figure 4.54: Session B4 final state *(B4:37:04)*

5

Conclusions

Let me summarize what I believe to be the contributions of this work. I have introduced two distinct products: the software artifact of Dynapad itself and this exegesis of it. Speaking generally, the software offers *practical* value and both products offer *theoretical* value. The nature of that value varies for different audiences: cognitive scientists, software designers, and end users. I will enumerate below the potential contributions toward each.

5.1 Dynapad as a Software Artifact

As a product, Dynapad has two facets. First, it offers particular *functionality*, which includes not only its intended features considered independently but also the overall cost structure which emerges as those features interact. The majority of this document has been devoted to examining that cost structure, and section 5.1.1 will summarize its potential value.

Second, that functionality is embodied in a particular *implementation*, which includes not only the actual lines of Dynapad's code but also the abstract structural relationships between its components. While the functionality sets up an activity landscape in which the user operates, the implementation imposes a similar landscape on the programmer by dictating the cost structure of *changing* the functionality. Except for a few implementation details in Chapter 3, this document has left that facet of Dynapad mostly implicit. Section 5.1.2 will consider its role.

5.1.1 Value of Functionality for End Users

The first contribution of this work is the potential for Dynapad's functionality to support end users in managing collections. However, I have never meant to suggest that Dynapad is in any sense the *best* application for collection management. On the contrary, I hope I've made clear that such a claim would be hopelessly undecidable for three reasons.

First, such a comparison assumes a utility function, as with Simon's crudest model, which operationalizes the value of a design into a single ordinal measure — an oversimplification I've deliberately avoided. Second, like any design, Dynapad makes trades: supporting certain facets of activity incurs costs elsewhere. My purpose is to understand these tradeoffs, as Dynapad manifests them. That is, *in what ways* is Dynapad effective and where do we pay for it? And third, comparison of different designs is never apples-to-apples: the style of collection management that Dynapad provokes is qualitatively different from that elicited by a different application. We might say that Dynapad provides the best support, in a trivial sense, for its activity because it's the *only* application which gives rise to exactly that activity. The activity we mean to support is a moving target, changing with each new design.

As I suggested in Chapter 1, the effectiveness of the design (reflected in question Q1) should ultimately be assessed by how that emergent activity fits into the ecology of surrounding practices. Such an evaluation, while important, is beyond the scope of this thesis. My purpose in this work is instead to shed light on Q2: *what is it about* Dynapad's design that gives it potential value?

With that purpose in mind, I will make a more modest claim:
Dynapad's design integrates a system of affordances that have demonstrated, here and in others' observations, important roles in managing both digital and paper collections, and the design reflects a considered effort to anticipate and accommodate the interactions between those affordances.

Specifically, those affordances include:

- Increased effective visibility and accessibility of collection items, achieved in three ways:

    1. by converting documents into portraits, making their *topical* distinctiveness

   visual and increasing the average information per pixel;

2. by supporting the unstacking of piles into open, de-occluded arrangements;
3. by reducing the cost of navigating, with an effective zooming and panning mechanism, to objects and views of interest.

- Support for implicit organization by reducing the hazards of inclarity and instability. Clarity can be improved by the expressive potential of open arrangements, enhanced by the visibility of individual items and the aggregation of groups. Stability is improved by insulating groups with containers and an unlimited supply of open space.
- Improved fluidity by supporting concurrent variants of organization. Specifically:
  1. Brushing of duplicates improves clarity by visually linking the scattered instances of items copied to participate in multiple spatial arrangements;
  2. Generalized brushing creates visual coherence for distributed groups and relations independently of spatial arrangement;
  3. Lenses allow automated arrangements independently of expressive manual positioning;
  4. Interactive history preserves alternatives of the entire workspace and allows *retroactive* valuation of exploratory arrangements.
- A generalizable protocol for adding localized automation, to supplement manual investment and bridge the gulf between high-volume fully-automated spaces and the flexibility and expressiveness of fully-manual control.

Dynapad as a Tool

In Chapter 1, I characterized Dynapad as a tool, an “outer” artifact with which the user creates another “inner” artifact of potential personal value. That inner artifact, the product of the user’s activity, is the organization invested into their workspace.

Above I’ve summarized the properties that make the tool effective in crafting its product. But so far, that assessment is like saying, for example, that a Widgetron has the right properties for making Widgets; it neglects the question of what that product is *good for.* As I've said above, ultimately that question must be answered by longitudinal observation of the product's role in the user's broader practices. But here I'll briefly summarize, in a qualitative description, the *kind* of value that Dynapad offers its user.

The Workspace as a Sensemaking Artifact

Let's focus on subject **A** since he, more than **B**, "finished" his product and reflected briefly on its value (lines 113–122 on page 199). By the end of his sessions, **A** clearly has a strong sense of his geography, and that sense has co-evolved with his artifact. That's sensemaking in action.

In the end, the artifact has value primarily as a focus for **A**'s knowledge. It wouldn't mean much to anyone else; the artifact cannot be stripped from the context of experience behind it. Furthermore, it's not clear that **A** needs the map to get value at that point. From his experience developing that map, he has made made extensive contact with his documents and built a conceptual framework for that collection.

So I would suggest that the "product" of personal value is not the material artifact, but the process of its development, the user's experience. Dynapad's greatest value is as a *catalyst for engagement.*

Imagine, as a thought experiment, emulating a physical analogue of Dynapad's basic "tabletop" functionality (see Section 3.5), which would afford most of **A**'s activity. Specifically, imagine putting **A** undisturbed in a large room containing only a photocopier and paper copies of his documents pre-arranged on the floor in the same initial grid as with Dynapad. The cost structure of that environment is similar but not equal to that of Dynapad; for example, long-range navigation is more expensive so wide-scale visibility is reduced. Therefore we should expect **A** to adopt somewhat different strategies and develop a somewhat different organization (e.g. perhaps more densely arranged). But it seems a reasonable speculation that **A** would produce *some* spatial organization and have the same basic relationship to it: his eventual arrangement would reflect the history of his sensemaking and shape his conceptualization of the collection. The product of his activity is likely to have the same kind of value to him as his finished Dynapad workspace.

What's the same in these parallel situations? Beyond the similarity of their cost structures, they both tacitly *invite immersion*: the immediacy of the documents and the room's insulation from external distractions draws the user into engagement with his collection. The setting offers the same motivational stance as playing a game: an implicit voluntary commitment to a restricted repertoire of action in an deliberately simplified environment.

Of course, even this mundane physical scenario still requires too much effort and resources (e.g. a personal ballroom) to enact. So one interpretation of Dynapad's value is not that it offers any special magic, but merely puts the user into the commitment of that room. It reduces to a plausible level the cost of entering that mode of commitment, and it is ultimately that mode and degree of engagement which catalyzes sensemaking, even in the absence of further computational magic.

But of course, Dynapad does offer further magic: brushing, lenses, stamps, etc. It puts *toys* in the room whose operations are always coupled to the collection materials, drawing the user into the collection instead of away from it. The value of these arrangement and visualization tools is not that they produce an objectively "correct" organization of materials but that they offer a cheap supply of stimulation to provoke fresh ideas and new ways of seeing.

In that light, my participants **A** and **B** were playing a game. Its "rules" are the invisible walls and paths of Dynapad's cost structure: barriers to some actions by their difficulty or subtlety, and beacons to others by their ease and apparency. Dynapad's value is the extent to which that game connects **A** and **B** with their materials. As a tool, its ultimate product is not the organization of the workspace per se, but the richness of the user's relationship to it.

5.1.2 Value of Implementation for Designers

The second contribution of this research is Dynapad's code, the *implementation* of its functionality.

Dynapad has served its purpose as a research prototype, but like many prototypes, it needs to be significantly reworked before being put to public use. Without going into detail, it requires three basic types of needed revisions:

Reinforcement: thorough debugging, testing, error trapping, and optimization;

Refactoring: cleaner separation of independent modules, restructuring of certain coordinated objects, and possible separation of client and server roles;

Translating to a more ubiquitous infrastructure, ideally web-based.

The last two categories especially require not just augmenting the existing implementation but aggressively redesigning and rebuilding it.

Therefore, Dynapad's code has value not as a finished product, but as an exploratory study. Despite its shortcomings, it represents an accumulation of lessons learned. A few of them I have articulated explicitly in these chapters, but many remain only in my head and can be elicited only by working further with the code, my own sensemaking artifact.

Let's appeal once again to the metaphor, outlined in Chapter 1, of the process of design as a two-level ecology: a user negotiating activity while developing and reacting to an artifact, all within a world which is itself under negotiation by the designer. As that designer, I play a role analogous to the user's role; I have negotiated not only the specific tactics of implementation, but also the very purpose of the research activity.

Programming as an Epistemic Activity

Implementing any design triggers *representational talkback* [73]. Even a non-functional mock-up or storyboard, for example, serves as a framework for eliciting, accumulating, and reconciling details, helping the designer work out the implications of design ideas. And a functional prototype is even more so; it must reckon with not only surface details but underlying processes. Therefore writing runnable code means facing decisions that could otherwise be deferred. Writing an algorithm makes visible specific choices and their alternatives. Programming is a forcing function for identifying — and inventing — the design space.

In my experience, the greatest challenge of that process is not in choosing, at any decision point, one alternative over another. Instead, the difficulty is in recognizing those decision points: acknowledging implicit assumptions as plans are made and remembering down the road where unexplored paths lie behind. Coding well means not only imbuing efficiency and flexibility, but also investing in cue structure to remind oneself of unexploited freedoms. It requires not only commenting effectively, but also choosing structures which reflect the problem space, even as they help to shape it.

It is only through that process that the problem space becomes defined. To paraphrase E. M. Forster: *I cannot know what I think until I see what I code.* That is, I cannot know what user behavior I hope to support until I make it possible with a sufficiently functional implementation. I cannot know the structure of the design space until I make certain specific decisions and, in so doing, expose others, while constantly trying to leave "hooks" to mark that trail of discovery.

In that light, the ultimate value of Dynapad's implementation is not as finished code, nor necessarily even as an abstract architecture. More fundamentally, that implementation serves as an implicit inventory of the decisions which have been exposed through the process of its development. Dynapad's code, however imperfect, has buried in it a geologic record of that negotiation. While much of that record remains implicit, this document is a first pass at exhuming it.

Dynapad's ultimate value, ironically, is that now I am ready to build Dynapad. Like all software, it is scaffolding for itself, in that it establishes conceptual structures needed to achieve it. Like the user's workspace, it has served me as a catalyst for *my* engagement with the cognitive phenomena of collection-management and of the subtle interplay between design and behavior.

But to describe this work as exploratory "scaffolding" is not to denigrate it. Instead, I mean to appreciate the inevitability and value of that exploration, to acknowledge the complex genesis of a complex ecology. Furthermore, that conceptual ontogeny is by no means unique to this research. I have given it special emphasis here not because that process has greater importance for Dynapad but because I believe it is universally important but largely under-appreciated in other work.

5.2 Value of the Analysis for Designers

The third contribution of this research is this document: the examination of how the details of Dynapad's design interact in a complex cognitive ecology which shapes the users' behavior.

This analysis is, at its core, a tale of two ecologies, one nested in the other.

Chapter 1 introduces that two-level framework for thinking about the process of design. In the outer domain, I, as designer, have negotiated both a landscape of technical constraints and opportunities and a changing conceptual framework of research and design goals. Dynapad, the product of that negotiation, is an inner environment whose details participate in shaping complex user behavior.

An environment for managing information (either digital or paper) is constrained by certain trade-offs, relationships of both reinforcement and inhibition between affordances. These constraints are reflected both in behaviors (e.g. piling vs. filing) that I and others [49, 70] have observed and in the design space for digital piling environments (e.g. [50]). Chapter 2 attempts a synthesis of some of these constraints, sketching a theoretical framework for interpreting and reconciling those observations. That framework is incomplete and perhaps inconsistent, but its value lies in *dissociating* certain affordances and facets of behavior which participate differently in the environment. This tactic is like splitting firewood: the resulting pieces may not yet be small enough for use, and many knotty chunks remain, but progress accumulates.

Chapters 3 and 4 apply that interpretive framework to Dynapad, exploring a network of interactions among the details of the design and the behavior they impel. The results demonstrate a familiar and sobering principle: *a design must be considered as a system.* Seemingly unrelated or insignificant aspects of the environment accumulate and interact to shape the cost structure in ways that significantly impact behavior.

All of the components of this narrative, like Dynapad's code itself, are *conceptual scaffolding.* They constitute a framework for thinking about Dynapad's design and interpreting observations of its use. Of course, this analysis is not definitive even for Dynapad, let alone of interfaces more generally. Instead, its value is as an example of what it means to expose the implications of a design.

As Dourish [18] and others have wisely observed, we shouldn't expect all scientific contributions to come packaged as tidy commodities, ready to install out-of-the-box into our own diverse contexts. Indeed, there are no products here that I expect others to apply in such a convenient fashion. My intent instead is to present, at an fertile level of description, a record of my own thinking, to give exposure to a collection of ideas that suggest potential. By no means have I solved the gnarled problem of digital information management. But I offer here an investment of chopping at it, in the hope that I and others find new facets to attack.

Bibliography

[1] Kiyoharu Aizawa, Kenichiro Ishijima, and Makoto Shiina. Automatic summarization of wearable video - indexing subjective interest. In *PCM '01: Proceedings of the Second IEEE Pacific Rim Conference on Multimedia*, pages 16–23, London, UK, 2001. Springer-Verlag.

[2] Deborah Barreau and Bonnie A. Nardi. Finding and reminding: file organization from the desktop. *SIGCHI Bull.*, 27(3):39–43, 1995.

[3] Dan Bauer, Pierre Fastrez, and Jim Hollan. Computationally-enriched 'piles' for managing digital photo collections. In *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing (VLHCC'04)*, pages 193–195. IEEE Computer Society, October 2004.

[4] Daniel Bauer. Personal information geographies. In *CHI '02 extended abstracts on Human factors in computing systems*, pages 538–539. ACM Press, 2002.

[5] Daniel Bauer, Pierre Fastrez, and Jim Hollan. Spatial tools for managing personal information collections. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS'05)*, page 104.2. IEEE Computer Society, 2005.

[6] Richard A. Becker and William S. Cleveland. Brushing scatterplots. In William S. Cleveland and M. E. McGill, editors, *Dynamic Graphics for Statistics*, pages 201–224. Wadsworth, 1988.

[7] Ben B. Bederson, James. D. Hollan, Ken Perlin, Jon Meyer, David Bacon, and George Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7:3–31, 1996.

[8] Benjamin B. Bederson. Photomesa: a zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*, pages 71–80, 2001.

[9] Benjamin B. Bederson and James D. Hollan. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 17–26, 1994.

[10] Jacques Bertin. *Semiology of Graphics.* University of Wisconsin Press, 1983.

[11] Hugh Beyer and Karen Holtzblatt. *Contextual Design.* Morgan Kaufman, 1998.

[12] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM Press, 1993.

[13] Matthew Brand. Understanding manipulation in video. In *FG*, pages 94–99. IEEE Computer Society, 1996.

[14] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of ACM Conference Human Factors in Computing Systems, CHI*, pages 181–188. ACM, April 1991.

[15] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.*, 9(2):99–122, 1991.

[16] Nils Dahlbäck, Arne Jönsson, and Lars Ahrenberg. Wizard of oz studies: why and how. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pages 193–200. ACM Press, 1993.

[17] Paul Dietz and Darren Leigh. Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226. ACM Press, 2001.

[18] Paul Dourish. Implications for design. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 541–550, New York, NY, USA, 2006. ACM Press.

[19] Paul Dourish, W. Keith Edwards, Anthony Lamarca, and Michael Salisbury. Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction*, 6(2):133–161, 1999.

[20] Susan Dumais, Edward Cutrell, JJ Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 72–79, New York, NY, USA, 2003. ACM Press.

[21] Adam Fass, Jodi Forlizzi, and Randy Pausch. Messydesk and messyboard: two designs inspired by the goal of improving human memory. In *DIS '02: Proceedings of the conference on Designing interactive systems*, pages 303–311, New York, NY, USA, 2002. ACM Press.

[22] S. Fertig, E. Freeman, and D. Gelernter. Lifestreams: an alternative to the desktop metaphor. In *Proceedings of ACM SIGCHI Conference*, pages 410–411, 1996.

[23] Scott Fertig, Eric Freeman, and David Gelernter. "finding and reminding" reconsidered. *SIGCHI Bull.*, 28(1):66–69, 1996.

[24] Matthew Flatt. PLT MzScheme: Language manual. Technical Report PLT-TR2006-1-v352, PLT Scheme Inc., 2006. http://www.plt-scheme.org/techreports/.

[25] David Fox. Composing magic lenses. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 519–525. ACM Press/Addison-Wesley Publishing Co., 1998.

[26] G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23, New York, NY, USA, 1986. ACM Press.

[27] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[28] Jim Gemmell, Lyndsay Williams, Ken Wood, Roger Lueder, and Gordon Bell. Passive capture and ensuing issues for a personal lifetime store. In *CARPE'04: Proceedings of the the 1st ACM workshop on Continuous archival and retrieval of personal experiences*, pages 48–55, New York, NY, USA, 2004. ACM Press.

[29] James Gibson. The theory of affordances. In Robert Shaw and John Bransford, editors, *Perceiving, Acting and Knowing: Toward an Ecological Psychology.* Erlbaum, 1977.

[30] Adrian Graham, Hector Garcia-Molina, Andreas Paepcke, and Terry Winograd. Time as essence for photo browsing through personal digital libraries. In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pages 326–335. ACM Press, 2002.

[31] Karen D. Grant, Adrian Graham, Tom Nguyen, Andreas Paepcke, and Terry Winograd. Beyond the shoe box: Foundations for flexibly organizing photographs on a computer. Technical report, Computer Science Department, Stanford University, 2003.

[32] D. Austin Henderson, Jr. and Stuart K. Card. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics*, 5(3):211–243, July 1986.

[33] Ron R. Hightower, Laura T. Ring, Jonathan I. Helfman, Benjamin B. Bederson, and James D. Hollan. Graphical multiscale web histories: A study of padprints. In

Proceedings of the Ninth ACM Conference on Hypertext, Mapping and Visualizing Navigation, pages 58–65, 1998.

[34] William C. Hill and James D. Hollan. History-enriched digital objects: Prototypes and policy issues. *The Information Society*, 10:139–145, 1994.

[35] Edwin Hutchins. *Cognition in the Wild*. MIT Press, Cambridge, 1995.

[36] Susanne Jul. Predictive targeted movement in electronic spaces. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 626–627, New York, NY, USA, 2002. ACM Press.

[37] Susanne Jul. *From brains to branch points: cognitive constraints in navigational design*. PhD thesis, University of Michigan, 2004. Chair-George W. Furnas.

[38] Susanne Jul and George W. Furnas. Critical zones in desert fog: aids to multiscale navigation. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 97–106, New York, NY, USA, 1998. ACM Press.

[39] H. Kang and B. Shneiderman. Visualizing methods for personal photocollections: Browsing and searching in the photofinder. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME2000)*, pages 15390–1542, 2000.

[40] Hyunmo Kang. Personal media exploration with semantic regions. In *CHI '03 extended abstracts on Human factors in computing systems*, pages 668–669. ACM Press, 2003.

[41] Hyunmo Kang and Ben Shneiderman. Mediafinder: an interface for dynamic personal media management with semantic regions. In *CHI '03 extended abstracts on Human factors in computing systems*, pages 764–765. ACM Press, 2003.

[42] David Kirsh. The intelligent use of space. *Artificial Intelligence(1-2)*, 73:31–68, 1995.

[43] David Kirsh. Metacognition, distributed cognition, and visual design. In Peter Gardinfas and Petter Johansson, editors, *Cognition, Education, and Communication Technology*. Lawrence Erlbaum, 2005.

[44] David Kirsh and Paul Maglio. On distinguishing epistemic from pragmatic actions. *Cognitive Science*, 1995.

[45] Scott R. Klemmer, Michael Thomsen, Ethan Phelps-Goodman, Robert Lee, and James A. Landay. Where do web sites come from?: capturing and interacting with design history. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8. ACM Press, 2002.

[46] M. Lansdale. The psychology of personal information management. *Applied Ergonomics*, 19(1):55–66, March 1988.

[47] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, 1(2):126–160, 1994.

[48] Peter Lucas and Lauren Schneider. Workscape: a scriptable document management environment. In Catherine Plaisant, editor, *CHI Conference Companion*, pages 9–10. ACM, 1994.

[49] Tom Malone. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. on Office Information Systems*, 1(1):99–112, 1983.

[50] Richard Mander, Gitta Salomon, and Yin Yin Wong. A "pile" metaphor for supporting casual organization of information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 627–634. ACM Press, 1992.

[51] K. Nakakoji, Y. Yamamoto, S. Takada, and B. Reeves. Two-dimensional spatial positioning as a means of reflection in design. In *Proceedings of ACM DIS'00*, pages 145–154, 2000.

[52] J. Patten and H. Ishii. A comparison of spatial organization strategies in graphical and tangible user interfaces.

[53] R. Pea, M. Mills, J. Rosen, K. Dauber, W. Effelsberg, and E. Hoffert. The diver project: Interactive digital video repurposing. *IEEE Multimedia*, 11:54–61, 2004.

[54] Jun Rekimoto. Timescape: A time-machine for the desktop environment. In *Proceedings of ACM CHI'99 Extended Abstracts*, pages 180–181, 1999.

[55] Jun Rekimoto and Eduardo Sciammarella. Toolstone: effective use of the physical manipulation vocabularies of input devices. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 109–117. ACM Press, 2000.

[56] George Robertson, Mary Czerwinski, Kevin Larson, Daniel C. Robbins, David Thiel, and Maarten van Dantzich. Data mountain: Using spatial memory for document management. In *Proceedings of ACM UIST'98*, pages 153–162, 1998.

[57] K. Rodden and K. Wood. How do people manage their digital photographs? In *Proceedings of ACM CHI'03*, pages 409–416, April 2003.

[58] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. The cost structure of sensemaking. In *Proceedings of the conference on Human factors in computing systems*, pages 269–276. Addison-Wesley Longman Publishing Co., Inc., 1993.

[59] Dan Saffer. Filepiles: Using a piling metaphor for digital document organization. Master's project, Carnegie Mellon University, School of Design, 2005.

[60] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–83, 1994.

[61] Eric Saund and Edward Lank. Stylus input and editing without prior selection of mode. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 213–216, New York, NY, USA, 2003. ACM Press.

[62] Robert W. Scheifler and Jim Gettys. The x window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.

[63] Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action.* Basic Books, 1983.

[64] Stacey D. Scott, M. Sheelagh T. Carpendale, and Stefan Habelski. Storage bins: Mobile storage for collaborative tabletop displays. *IEEE Comput. Graph. Appl.*, 25(4):58–65, 2005.

[65] Stacey D. Scott, M. Sheelagh T. Carpendale, and Kori M. Inkpen. Territoriality in collaborative tabletop workspaces. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 294–303, New York, NY, USA, 2004. ACM Press.

[66] Abigail Sellen and Richard Harper. *The Myth of the Paperless Office.* MIT Press, 2001.

[67] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.

[68] Herbert A. Simon. *The sciences of the artificial (3rd ed.).* MIT Press, Cambridge, MA, USA, 1996.

[69] Walter F. Tichy. Design, implementation, and evaluation of a revision control system. In *ICSE '82: Proceedings of the 6th international conference on Software engineering*, pages 58–67, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

[70] Steve Whittaker and Julia Hirschberg. The character, value, and management of personal papaer archives. *ACM Transactions on Computer-Human Interaction*, 8(2):150–170, 2001.

[71] Allison Woodruff, Andrew Faulring, Ruth Rosenholtz, Julie Morrsion, and Peter Pirolli. Using thumbnails to search the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 198–205. ACM Press, 2001.

[72] Dev Yamakawa. An ethnographic study of image organization. Unpublished Honors Thesis, UCSD, 2003.

[73] Y. Yamamoto, S. Takada, and K. Nakakoji. Representational talkback: An approach to support writing as design. In *Proceedings of the 3rd Asia Pacific HCI Conference*, pages 125–131. IEEE Computer Society, 1998.

[74] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the conference on Human factors in computing systems*, pages 401–408. ACM Press, 2003.