A Better Mythology for Computing

Jed Harris, Austin Henderson
Pliant Research

This is an extended version of a talk given at SigCHI, May 18, 1999.

Computing is becoming constitutive of social groupings

Examples

- Financial systems
- Telecommunications
- Utilities (e.g. power grids)
- Airlines
- Distribution

So, limitations in computing are becoming limits for society

When our computer systems start to limit social evolution, we had better think hard about evolving our computers.

Social groupings need:

To be *responsive* to local needs
- Deal effectively with unanticipated demands

To be *coherent* across larger regions
- Coordinate disparate activities so they reinforce each other

To *scale*
- Extend to support great...
  - **breadth**: many "compatible" instances
  - **depth**: many diverse, integral elements

Responsive:

improv theater,

effective political campaigns

Coherent:

symphony orchestras,

telephone systems

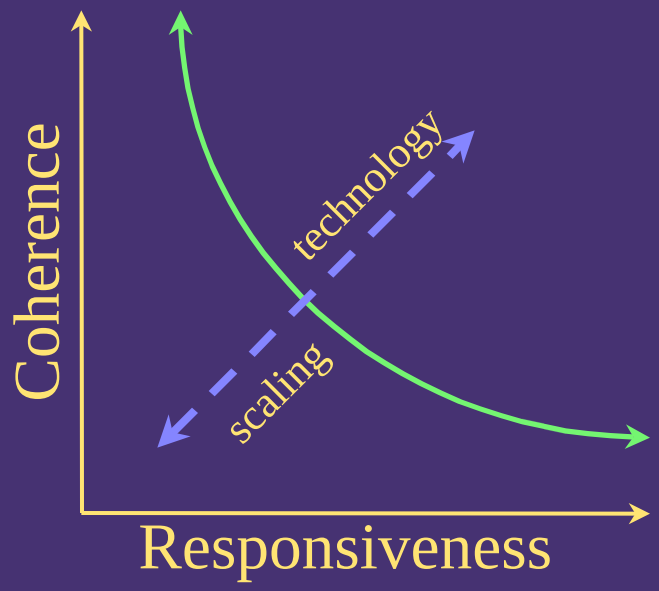
Large scale:

MacDonalds,

World Wide Web,

international finance

Responsiveness vs. coherence



In a given system, as responsiveness increases, coherence tends to decrease, and vice versa—a classic tradeoff.

Scaling makes this tradeoff sharper. As systems get larger, they have to work harder to maintain their coherence, and this increasingly makes them unresponsive. Conversely, large systems that allow great local responsiveness (such as the World Wide Web) have difficulty maintaining coherence.

This tradeoff, like most others, can be improved though the use of appropriate technology. Such technology offers the possibility of both increased coherence and increased responsiveness, even as systems grow large.

Pliant research is focused on developing technology to improve this tradeoff.

Example: Responsiveness vs. Coherence*

Situation: Clerk is filling in “ship-to address”

Coherence: System validates address, selects optimal warehouse, automatically prints bar coded label for UPS at ship time

Unanticipated demand: Copier is on a barge

Responsive improvisation:

- enter “Call Bob: 211-555-1234”
- but extreme coherence makes this impossible!

* Fikes and Henderson, after Eleanor Wynne

When this example was originally described, the clerk could improvise an entry on a paper form, or perhaps call the guy on the shipping dock, and get the supplies to the barge.

However, as computers become constitutive, the data entry validation, automatic order routing to the optimal warehouse, and integration with computerized shipping procedures makes such improvisation impossible.

Problem: Now, computers maximize coherence

This greatly impairs responsiveness

Results

- systemic rigidity and fragility
- user burdens
- unhealthy pressures on social processes

So...

When most computing systems encounter internal inconsistency, they simply crash. Designers try to avoid this by giving their systems a single, consistent perspective on the world.

Users end up bearing the burden of making the world look the way the systems demand, to insulate them from the messy reality that they can't handle.

Increasingly, social systems are being forced to operate in ways that are conveniently supported by computers, rather than the other way around.

Luckily some important types of systems, such as internet routers, can handle some messiness in their interaction with the world. This is part of the reason that the internet, and especially the web, have grown so rapidly.

Roadmap

Tension between computing and society

➳ Roots of the tension: current design mythology

A better mythology

Two research agendas:

- Extending current computing
- Developing a new type of computing

What can we do?

explore the roots of the problem

adopt new design and research approaches

improve the tradeoff

Current design mythology: "Total consistency"

The stories we tell about our practice

Standard ideology in "core" computer science

- Protocols, languages, applications, data bases, operating systems, good old fashioned AI

Maximizes coherence, ignores responsiveness

This is a **choice;** we can choose differently

- how did we make this choice?

Computer theorists tend to get upset when we suggest that we have to step back from total consistency. Without total consistency, existing models for computer semantics fail.

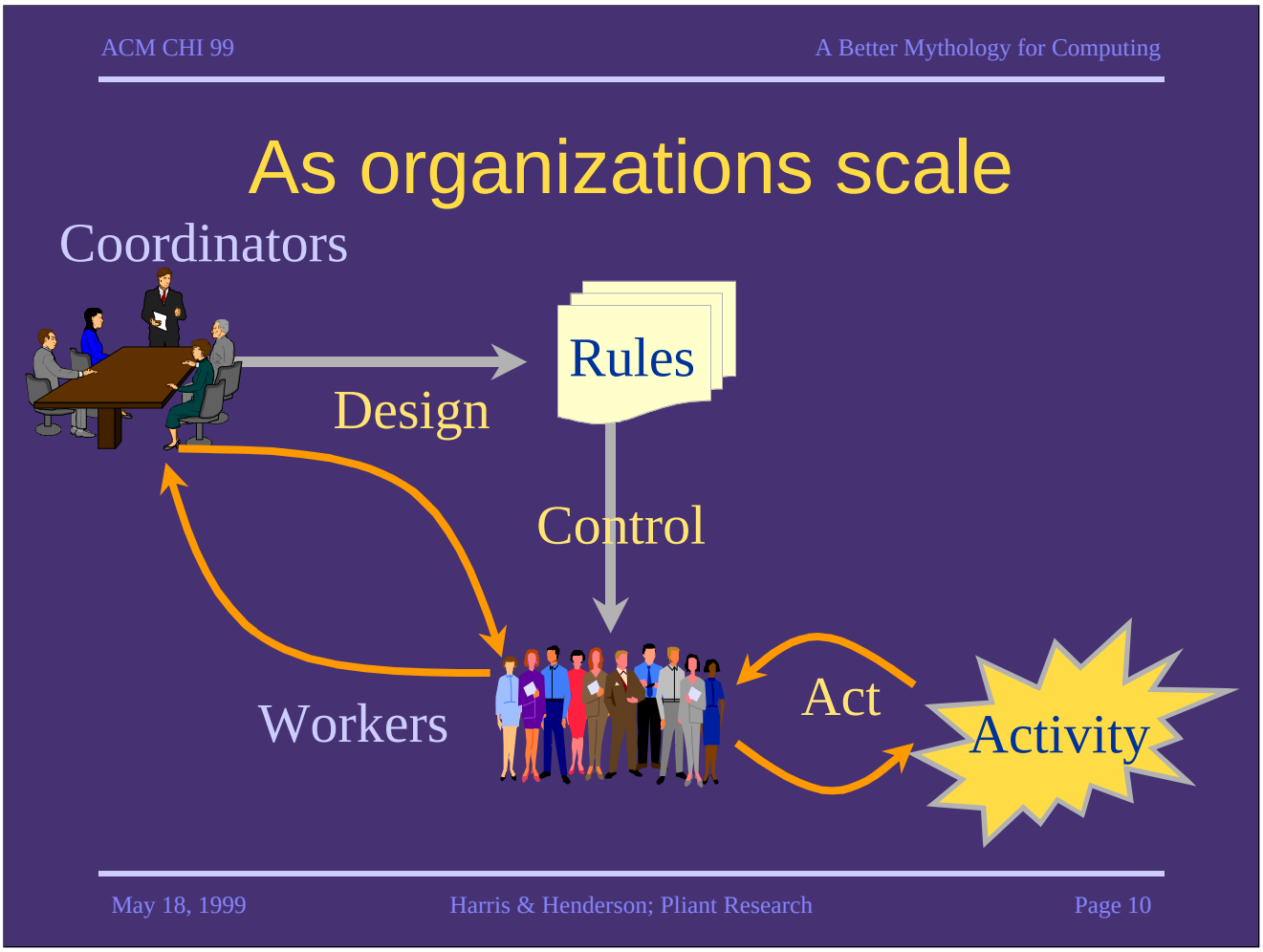
In fact no **real** system is totally consistent. However, this focus on the ideal of total consistency has left us with no theoretical tools for understanding real systems.

Instead, our theories have defined real systems as approximations to ideal, totally consistent systems. This has produced absurd results in areas such as linguistics, cognitive science, artificial intelligence, and user interface design.

It's time to recognize that this is a dead end and look for a better approach.

All real organizations work because of feedback loops between the coordinators, the workers, and the activity they carry out in the world.

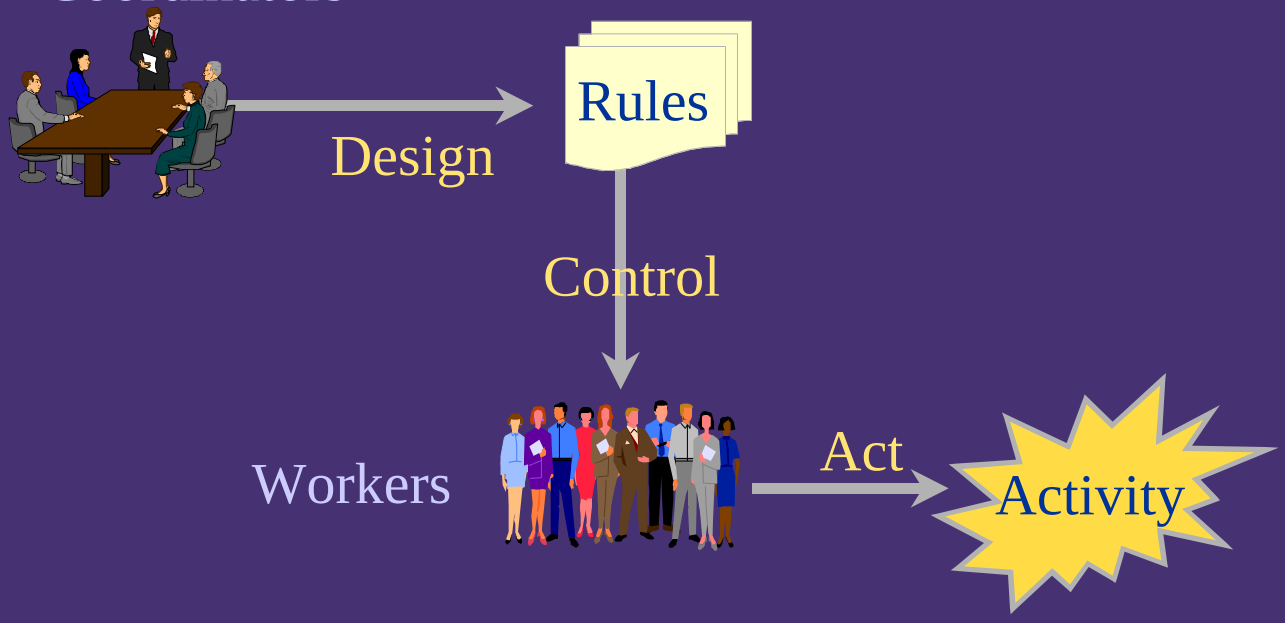
(Of course, in real organizations the coordinators and the workers overlap and are often the same people. In separating them we have already made a concession that we will ultimately have to take back.)



As soon as human groups get large, explicit control systems and rules emerge. Explicit rules probably arise as soon as human organizations get large; codes of law show up eve in very early cities (e.g, the Code of Hammurabi,, ca. 1750 BC).

Even mathematics has its roots in the need for coordination in large groups; it initially arose from recipes for coordinating large construction projects such as the pyramids, adjudicating disputes about land ownership, and tracking large inventories of goods.

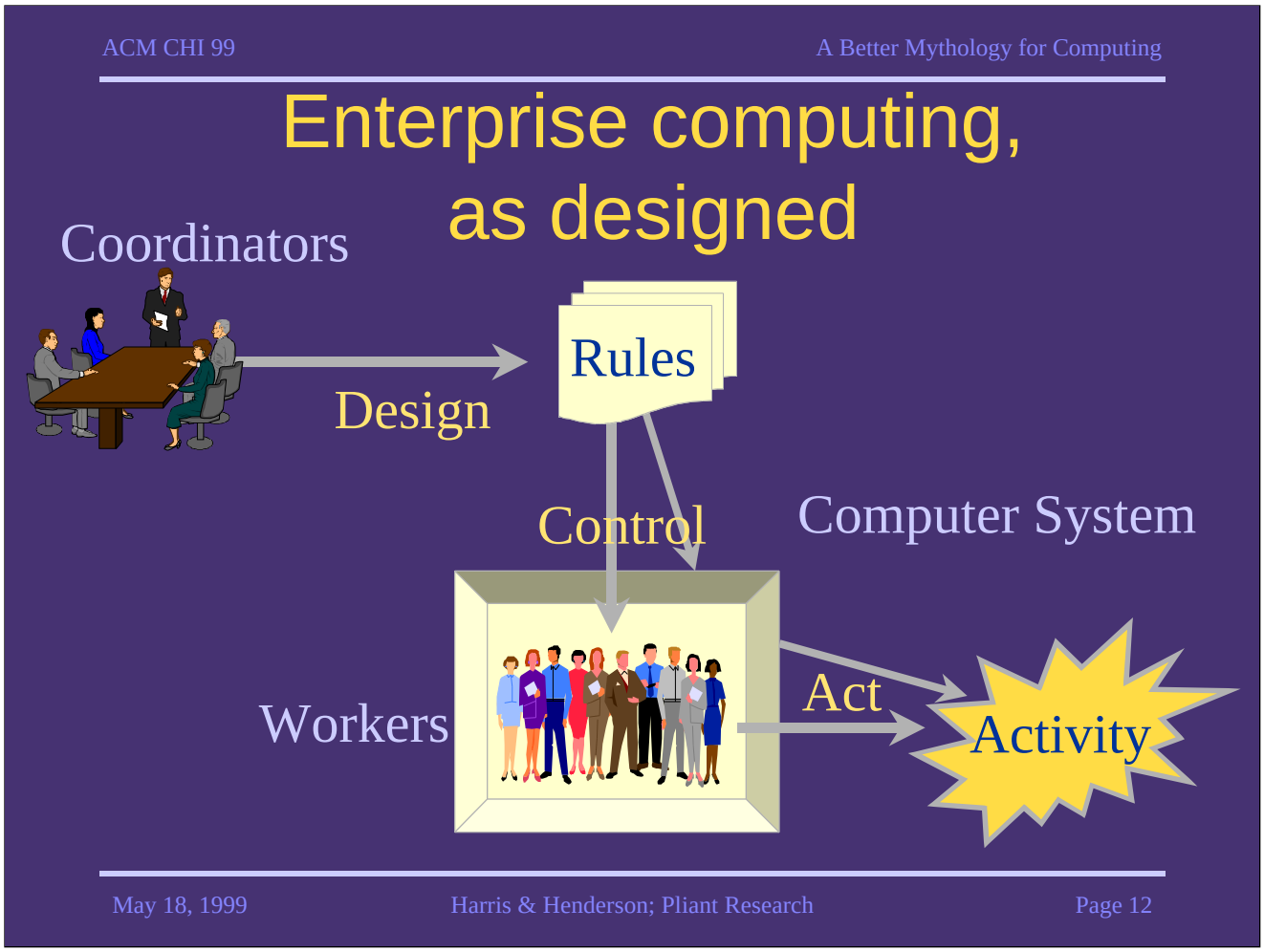
Bureaucratic mythology



The feedback loops that make organizations work are implicit, spontaneous, and far too complex to analyze in detail without modern tools such as video cameras and game theory.

The rules of an organization, on the other hand, require explicit formulation and enforcement.

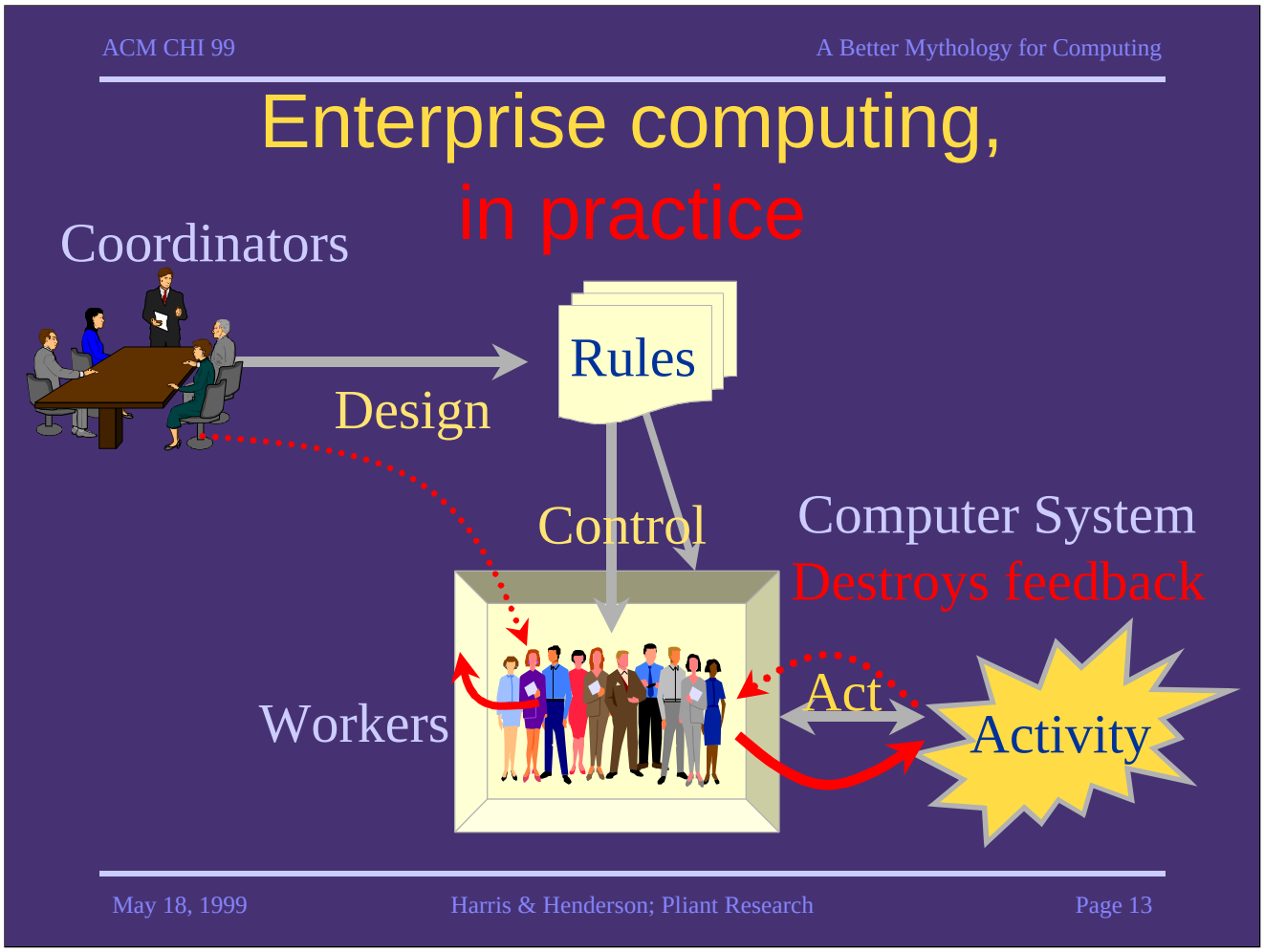
As a result it is easy to ignore the feedback loops, and come to believe that the rules, and the hierarchy that enforces them, **are** the structure of the organization.



Computer systems were designed to automate rule-following behavior: the work of human “computers” calculating ballistics tables, maintaining financial accounts, and so on.

They were developed at the intersection of several intensely rule-oriented cultures: military bureaucracies, mathematicians, and large, engineering-intensive corporations, such as telephone companies and office equipment manufacturers.

Naturally, the core ideology of computer system design is totally permeated with the assumption that computers are rule-following machines, and more generally, that all human activities can and should be described in terms of a consistent set of rules.



Unfortunately the idea that all human activities can be described in terms of a consistent set of rules fails to take into account the feedback loops that actually make organizations work, and the constant negotiation that these loops entail.

As a result, computing systems tend to break those loops. The more effectively the computer systems control the activity, the more completely they break the loops.

The resulting problems are invisible within the consistent perspective of the computer system and so people have to bear the brunt of patching them up, and usually have to fight the computer system to do it.

And scaling makes it worse: worlds in collision

Even if a system can be made more or less workable, big problems emerge as soon as it has to be integrated with other systems.

When banks merge, for example, millions of dollars (sometimes hundreds of millions) can be lost, and careers destroyed, because of the difficulty of integrating the systems.

Pliant vision

Computing that enhances and benefits from human creativity and diversity

instead of trying to suppress them

Most existing computing technology seems to be directed at either carrying out menial tasks efficiently, or keeping people in line.

It's time to develop computing that can work with people at a higher level.

Roadmap

Tension between computing and society

Roots of the tension: current design mythology

➾ A better mythology

Two research agendas:

- Extending current computing
- Developing a new type of computing

Now let's see how we can realize this vision...

Non-Solutions

Classical AI

- would a sufficiently large, complex, perfect bureaucracy be responsive?

Adaptive systems

- neural networks, hidden markov models, etc.
- can add “rubber bumpers” to rigid systems
- how will these technologies handle a large mission critical application?

The approach promoted by Artificial Intelligence from the 1940s through the 1980s (and still supported by many researchers today) is a striking example of how very intelligent people can adopt fruitless beliefs because of the core ideology of computer science. Why else would one think that relatively simple rule-based systems with no ability to learn should be able to emulate human beings…

Adaptive systems, while they are useful today (for example in speech and handwriting recognition) don’t currently scale. The big stumbling block is composition. Rule-based architectures provide very strong mechanisms for composing larger systems out of smaller ones. So far, we haven’t found such mechanisms for adaptive systems. Without composition mechanisms, for example, we can’t **explain** anything to an adaptive system...

A better mythology: "Dynamic engagement"

Balance responsiveness and coherence

- honor and support "self-control"

Scale without total consistency

- find other ways to sustain coherence

Work with richer ontologies

- outside the focus of logic and "classical" computer science

Engagement mixes cooperation and contention. In a dynamic engagement the rules are always being negotiated.

The focus of pliant research is to improve the tradeoff between responsiveness and coherence, especially as we build large systems.

In addition, we need to recognize how rich our ordinary experience of the world is, compared with the types of semantics currently used in computer science.

Richer ontologies

Multiple concurrent perspectives
- no privileged/definitive viewpoint

Often inconsistent, vague and equivocal

Often mutually incommensurable
- potentially incommensurable “all the way up”
- but if they interact, not “all the way down”

Drifting with respect to each other

These are ordinary facts of life in human interaction, but they are denied or explained away in essentially all current mathematical theories of semantics. As a result, we have no way to deal with them in our computer systems.

To begin with, it is essential to recognize that these **are** facts of life, and that computer system designers have to deal with them somehow.

In the longer run we must develop techniques that let us fully support and honor rich ontologies.

Roadmap

Tension between computing and society

Roots of the tension: current design mythology

A better mythology

➳ Two research agendas:

- Extending current computing
- Developing a new type of computing

The pliant argument may not convince many people with a strong investment in the status quo.

Instead, we need to focus on building systems that just work better than ones limited to total consistency.

These research agendas are focused on building systems that work better.

Becoming Pliant

Design focus: Pliant use of rigid systems

Not rocket science;
Let's do it!

System focus: Create pliant computing

Rocket science;
Research needed!

There is actually a spectrum of approaches to implementing pliant systems, from very near term to very long term.

In this presentation, we have chosen to describe approaches at the ends of the spectrum, but we are also actively exploring intermediate options.

Design focus: Pliant use of rigid systems

We can start without exotic technology

- find lots of "low hanging fruit"

Five types of examples...

We believe it is possible to take existing systems and extend them to support pliant usage. This only takes us so far, but it is a good start.

We have found that it is easy to generate good examples. When we look at an existing computing system, and the practices of people who use it, we can usually find ways to make it more pliant and better at supporting those practices.

1. Capturing unexpected data: Margins on forms*

Formal: Manage jobs via computer system

Informal: Actual job emerges in the doing

- annotations in margins of **paper** job sheet

But: No "margins" in computer database

- store paper in parallel with computer records
- reverse engineer right "story" for computer

So: Provide "margins" on computer forms

- **Users**: avoid parallel filing, easier retrieval
- **Designers**: better view of user practices

*Bowers, Button and Sharrock

This example is based on studies of print shops that were "improved" by adding a computerized scheduling and billing system.

The computer system couldn't reflect the real variation and complexity of the workers actual practices, such as splitting jobs across machines, using special paper, suspending one job to do a more urgent one, and so forth.

As a result, the workers had to use the old system to keep the print shop running. Then they had to figure out how to lie to the computerized system to generate the right records.

At a minimum, "margins" would have made the workers' jobs easier, because it would have helped them maintain their internal feedback loops.

If the computer system designers used the information that ended up in the margins, they could have extended the system to better support the print shop practices, and produced better records for management at the same time.

2. Using unexpected data: Invalid field values*

Formal: Ship-to address field (validated)

Informal: Copier is on a barge

- address only known at shipping time
- enter “Call Bob: 211-555-1234”

But: System accepts only “valid” addresses

- shipping process may be automated

So: Allow “flagged” values, ask users for help

- **Users**: easier communication, grow conventions
- **Designers**: better view of user practices

* Fikes and Henderson, after Eleanor Wynne

Here is the “copier on a barge” example that we saw earlier.

Even if the system is fully automated, we can still let people enter “invalid” data (perhaps after the system warns them that it can’t understand).

Then we need to have the system “call for help” from humans when it encounters data it can’t process. So in this case when the order is ready, the system would call for help from a human in the warehouse.

Once we’ve added this path, the computer system actually helps the people in the organization communicate effectively, rather than getting in their way.

Such adaptations aren’t trivial, but they aren’t technically challenging. The main reason they aren’t common is the “rule bound” mentality in system design.

3A. Ecological software growth: Buttons*

"Scriptable" buttons embedded in documents

- easy to use—invoke by clicking
- easy to exchange—mailable
- easy to view and mutate—small and simple

Observed use

- users needed "permission" to "play" ("Whatever is not required is forbidden.")
- users and designers crystallized regularities

*an InterLisp technology at Xerox RC Europe - Cambridge

The users chose and propagated buttons they found useful, so crystallization happened through reproductive success in the user environment. Users rarely changed the buttons themselves.

At the same time the designer noticed how buttons were used most frequently, and what users requested most often. He extracted what he could see as regular, and built it into the Buttons framework. As a result the buttons got a lot easier to adapt through external controls.

Eventually the button framework was general enough to satisfy most users' requirements, and change slowed (but did not stop).

3B. Ecological software growth: Web content

Source is always inspectable

- easily understood map from source to image
- easy to copy and mutate
- easy and safe to experiment

Observed use

- web page innovations propagate quickly
- local design continually stretches standards
- standards-makers can survey design population
- standards chase designs but never catch up

The web development and web standards communities recognize the importance of the “View Source” command in all web browsers, which allows users to see how effects are created.

Discussions on changes to browsers and standards often turn on what web page designers are doing with existing techniques. Participants often survey existing practices in one way or another.

However, as far as we know, this ecology of content hasn’t been studied explicitly **as** an ecology. It seems like a pretty clear example of “memes”, in this case identifiable because they are recorded in HTML XML, and scripting languages.

4. User feedback on design: Collaborative documentation*

Formal: Documentation comes with system

- generic and typically lags user practices

Informal: Users help each other in context

But: Finding good practices is hard and slow

So: Help users share practices through networked help system

- **Users**: more timely, relevant help
- **Designers**: better view of user practices

*Julian Orr; Danny Bobrow; Tom Malone; Henry Lieberman

In many cases, people actually solve their problems by asking questions on Usenet newsgroups. These questions and answers are often then crystallized into a “Frequently Asked Questions” list (FAQ). In effect, people are cobbling together a collaborative documentation mechanism.

There are various experimental systems that approximate networked help. However, as far as we know nothing is under commercial development that allows users of a program to post inquiries when their help system doesn’t answer their questions, and which then melds the answers back into the basic help system.

The most relevant commercial systems are “knowledge bases” for telephone help desks. These could probably be used as the basis for networked help systems reasonably easily.

5. Social coherence control: Large development projects*

Formal: Specs, schedules, ownership, etc.

Informal: Social interaction is real control

But: Need system support for social process

- to support asynchronous distributed work
- to provide outside visibility and auditing

So: Bug databases, weak serializability, etc.

- **Users:** social processes become more scaleable
- **Designers**: for now, users **are** designers…

*Becky Grinter

This sort of mixed computer and social control mechanism for software projects has been around for over a decade, but it has gotten a lot more visible and important with the emergence of huge networked Open Source projects (such as Linux and Apache) in which the participants rarely, if ever, meet face to face, and in which the management of the project is done almost entirely through the software systems.

Significantly, in these Open Source projects, there is no strong management hierarchy or organizational boundaries. The project is volunteer run and self-organized. Never the less, these projects have created products that dominate their markets.

As with the evolution of web idioms, the nature of the medium would make longitudinal studies of the process relatively easy, since most of the interaction is captured automatically.

Becky Grinter, “Recomposition: Putting it All Back Together Again” Proc CSCW 98, Seattle WA, ACM, New York. Pp 393-402

Coordinators

Design

Not rocket science;
Let's do it!

Control

Computer System

Workers

Act

With only a moderate effort, we can restore and even enhance the broken feedback loops that support organizational health, while still using computers to provide improved organizational scalability.

This effort would also give us valuable insight into the practices that actually maintain the fabric of organizational coordination, but which are largely invisible today.

As we improve our designs, we will be developing the social and technical understanding and acceptance that will be needed for a more deeply pliant technology.

This is good, but **not enough**!

Design focus gives **users** more pliant ontologies

But **computers**

- internally still depend on total consistency
- so they are still limited to **rigid** ontologies

So users are still doing all the ontology work

No matter how well we rework rigid systems to ameliorate these problems, they are still present.

As our systems scale and become more essential to our organizations, the tradeoff between coherence and responsiveness will get more painful, and the burdens on people in these organizations will continue to increase.

We must attack this tradeoff at a deeper level.

Roadmap

Tension between computing and society

Roots of the tension: current design mythology

A better mythology

Two research agendas:

- Extending current computing
- ⇛ Developing a new type of computing

We have looked at the sort term option. Now let's explore the other end of the spectrum: the longest term options we can understand well enough to research.

System focus: Create pliant computing

Four potential aspects

- Alexandrine patterns
- Enacting patterns
- Collaborating activities
- Evolving system

Rocket science; Research needed!

Using

- continuous values, field-like interaction
- probabilistic (rather than logical) inference
- thick scenes rather than thin descriptions

This description of this option is somewhat impressionistic because much of the content must be determined by the research itself. However we believe that the description captures essential elements that will be needed in some form.

We have explored the underlying technology (mentioned under “Using” above) but did not have space to discuss it in this presentation.

Continuous values and probabilistic inference are closely related, and give us the basic semantics needed for rich ontologies.

The contrast between thick and thin is exactly the ethnological distinction first described by Clifford Geertz, which turns out to apply very well to the design of complex software. Current programming ideals (modularity, clean architecture, etc.) lead to very thin descriptions. Pliant systems would be much more internally redundant, partially contradictory, multiply interpretable, and thus thicker.

1. Alexandrine patterns

Build the whole system out of patterns, as described by Christopher Alexander:

- "a swirling intuition about form… a fluid field of relationships"

Pattern characteristics:

- inherently metaphorical: used by "seeing as" or "reading as/on"
- express a problem and its solution—i.e. values
- interact through forces not predicates

Christopher Alexander invented patterns for human communication (about building). They are composable and form a language. However they are **not** symbolic or syntactic in the sense that language is; they have topological and analogical relationships to the structures they describe, and they can overlap and intertwine in complex ways.

Patterns also have an inherent "grain" that is much stronger than typical programming language constructs (but perhaps comparable to natural language words or phrases). A pattern helps users judge whether a given instance is "better" or "worse" according to the pattern's inherent values.

In the design process, patterns "push" and "pull" their underlying material toward better fit. Since patterns overlap, multiple patterns may be pushing or pulling on the same underlying features. This process might lead to a design that beautifully satisfies all the values, or it might get stuck in an unsatisfying local optimum. Such a local optimum typically takes a creative leap to escape.

2. Enacting patterns

Grasp a situation in terms of patterns, and thereby change it

Enaction characteristics

- observable, redirectable ⇨ manageable
- can always "call for help" (mixed initiative)
- can have conflicts and creative resolutions
- can use history as a resource
- typically not deterministic

We see patterns as being enacted: applied and elaborated incrementally. We see this enaction as open to inspection and intervention by users and designers—more like building a house over time than transitioning instantaneously between states in a finite automaton.

Enaction of patterns may have conflicts, get stuck, need help, and it creates history as it goes. This history then becomes a resource: when we got into similar situations in the past, what did we try, what worked, and what didn't work?

However, like life and evolution, the enaction of patterns isn't entirely predictable or repeatable. If we rewind the tape and run it again, we won't get exactly the same result. How close we come to the same result is mainly a function of how good our solution was compared with the alternatives, and how rough the landscape of alternatives is.

3. Collaborating activities

Organize massively parallel diversity to maintain the right amount of coherence

- monitor and manage coherence as a system resource like storage, bandwidth, or CPU time

Activity characteristics

- interact through enaction
- will often have incommensurable “languages”
- can always find common ground by going “down” (finer grain, more specific cases)

The description of enaction gives us a rough idea of how we would carry out specific “computational” processes. Now we need to consider how such processes would interact.

Processes may mesh well, and enhance each other’s effectiveness. In that case, they will tend to produce mutual coherence and we don’t need to worry.

On the other hand, processes may find themselves in a “tug of war” over the underlying ontology, trying to define shared elements in different ways, and reducing coherence. To advance its values, each process will need to search for alternative perspectives that can reduce the conflict. Through mutual search the processes may find compatible perspectives that increase coherence.

We need to develop ways to monitor and manage this sort of interaction, and more generally to treat coherence as a system resource, like memory, CPU time, or network bandwidth. Similar problems have been addressed, for example, in controlling thrashing in processor scheduling.

4. Evolving system

Manage large-scale change and growth by selecting and integrating local change

System characteristics

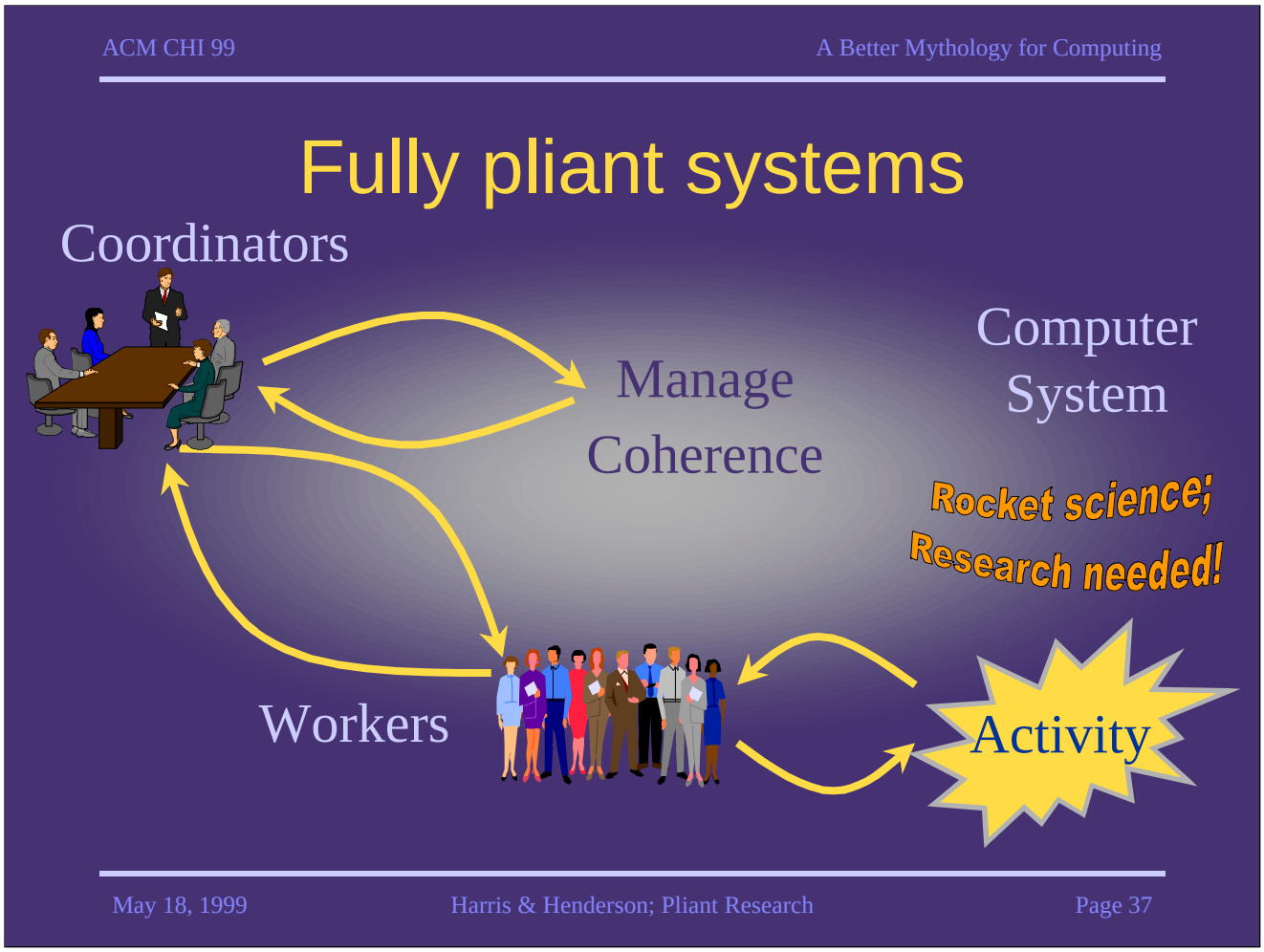
- keep on all the time
- revise locally, incrementally “on line”
- filter, consolidate and propagate local changes to maintain large-scale coherence
- let different parts drift in different ways

At the highest level, we need to recognize the nature of large-scale systems from now on: such systems cannot be built and rebuilt, they can only be evolved.

Today’s large systems, for example those which support enterprise-wide services or networked e-commerce, cannot be developed or modified “off line”; they are always on, and must be developed and modified incrementally on a “service maintained” basis.

This inherently requires the ability to change different parts of the system somewhat independently, to introduce and then resolve inconsistencies without ever rendering the system unusable.

In other words, such systems require pliant technologies.



In this final view, we summarize the long-term potential of pliant computing. The original feedback loops that make organizations work are restored. In addition, the rules and the computer box are gone. Instead of these rigid structures, computer systems have evolved into a pervasive, dynamic backdrop for organizational activity that allows explicit management of organizational coherence.

We believe that computer systems can change from an iron cage limiting flexibility and creativity to a framework that enhances both organizational strength and agility.

Pliant Research

www.pliant.org

Jed Harris
jed@pliant.org
510-524-4350

Austin Henderson
henderson@pliant.org
650-747-9201

Slides online at http://www.pliant.org/talk-7-99.pdf

Please get in touch with us if you have questions or want to discuss these ideas further.