19. Information Architecture

INFO 202 - 31 October 2007

Bob Glushko

Plan for Today's Lecture

Broad and narrow definitions of Information Architecture

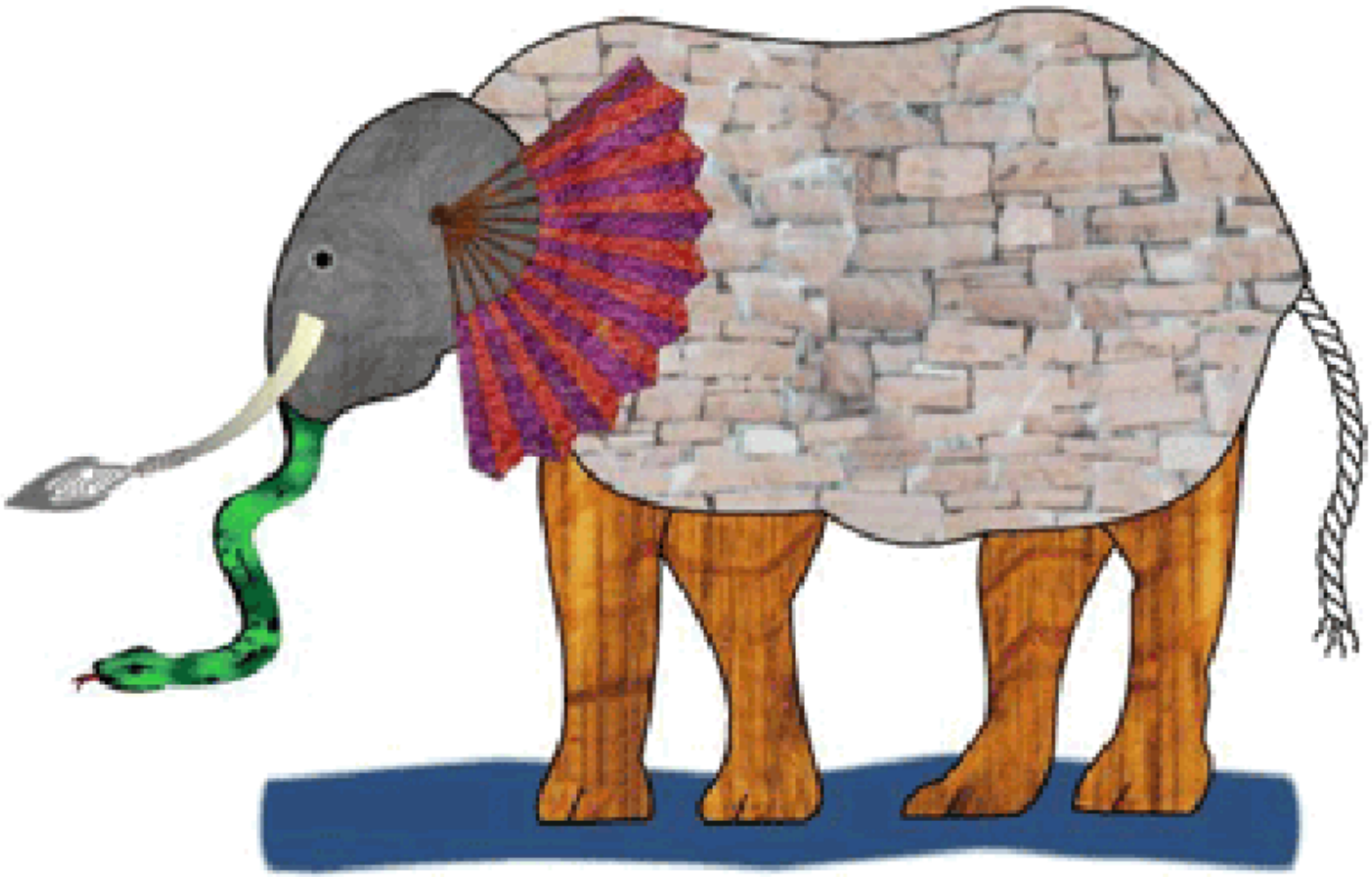
Model-based user interfaces

Principles for Information Architecture

- Separation of content from structure and presentation
- Structuring principles
- Reinforcing structure with presentation
- Internationalization and localization

User interface design patterns

Defining "Information Architecture" [1]



Defining "Information Architecture" [2]

Most people would include:

- Content organization / data modeling
- Navigation / interaction design

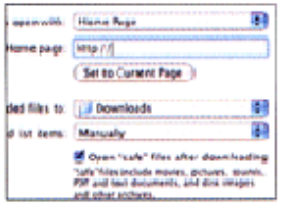
Some people would also include:

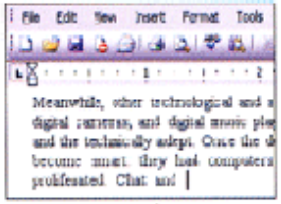
- Visual / graphical design

Some even include:

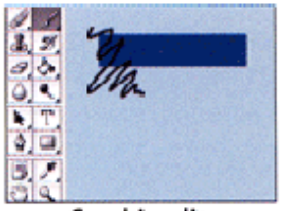
- "Experience" design
- "Virtual environment" / "mediated realities" design

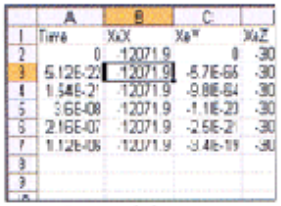
User Interface Design Idioms (Tidwell)


Forms

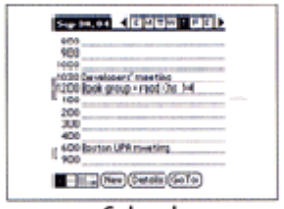


Text editors

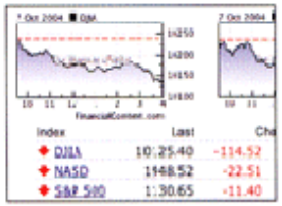

Graphic editors


Spreadsheets


Browsers


Calendars


Media players



Information graphics

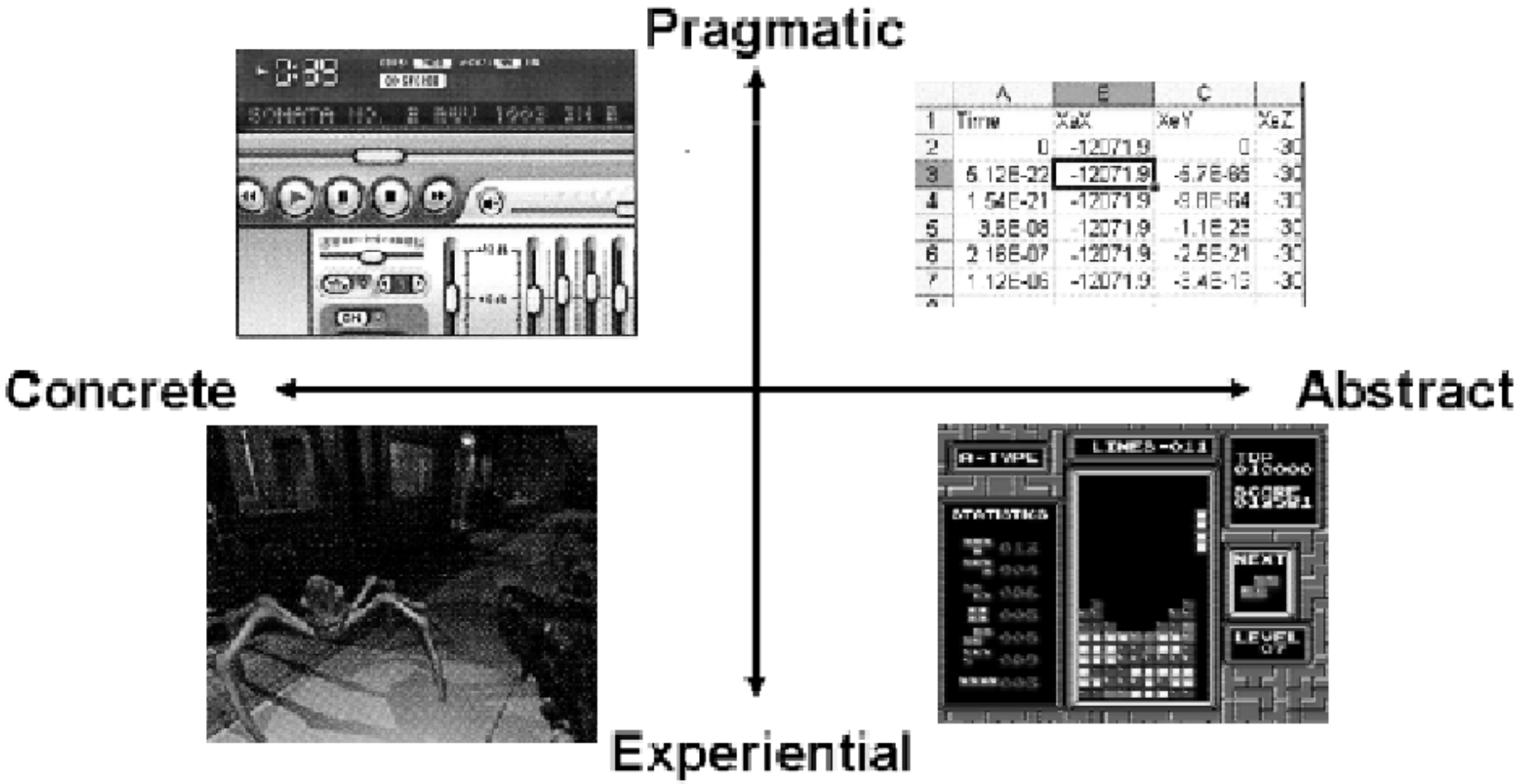

Immersive games

Web pages


Social spaces

E-commerce sites

How Broad a Scope for Information Architecture?



Implications of a Broad Scope

The broader the scope of contexts to which "Information Architecture" applies, the fewer design principles and methods will apply to all of them

Those that apply will necessarily be qualitative and heuristic

Quality designs can emerge, but quality can't always be predicted

There will be little reuse of design patterns or components

Few or none of the design and implementation activities can be automated

Methodology for Broadly-scoped IA

1. Requirements gathering and analysis
2. Development of "personas" -- detailed and personalized depictions of exemplars of user types
3. Prototyping / wireframing with static information sources
4. Iterate these steps until you run out of time or resources
    - Heuristic evaluation
    - Usability testing
    - Revise prototype (incremental addition of functions, features, and "business rules")
5. Connect to "live" information sources and deploy

Nielsen's Ten Usability Heuristics

Much IA is conducted using qualitative and heuristic guidelines

A classic set is Jakob Nielsen's
http://www.useit.com/papers/heuristic/heuristic_list.htm

- "The system should speak the users' language, with words, phrases and concepts familiar to the user"
- "Follow real-world conventions, making information appear in a natural and logical order"
- "Follow platform conventions"
- "Dialogues should not contain information which is irrelevant or rarely needed"

These are certainly helpful, but they assume significant expertise and are not always consistent with each other

A Narrowly-scoped IA?

Can we narrow the scope of "Information Architecture" to make its methods and principles more rigorous and deterministic?

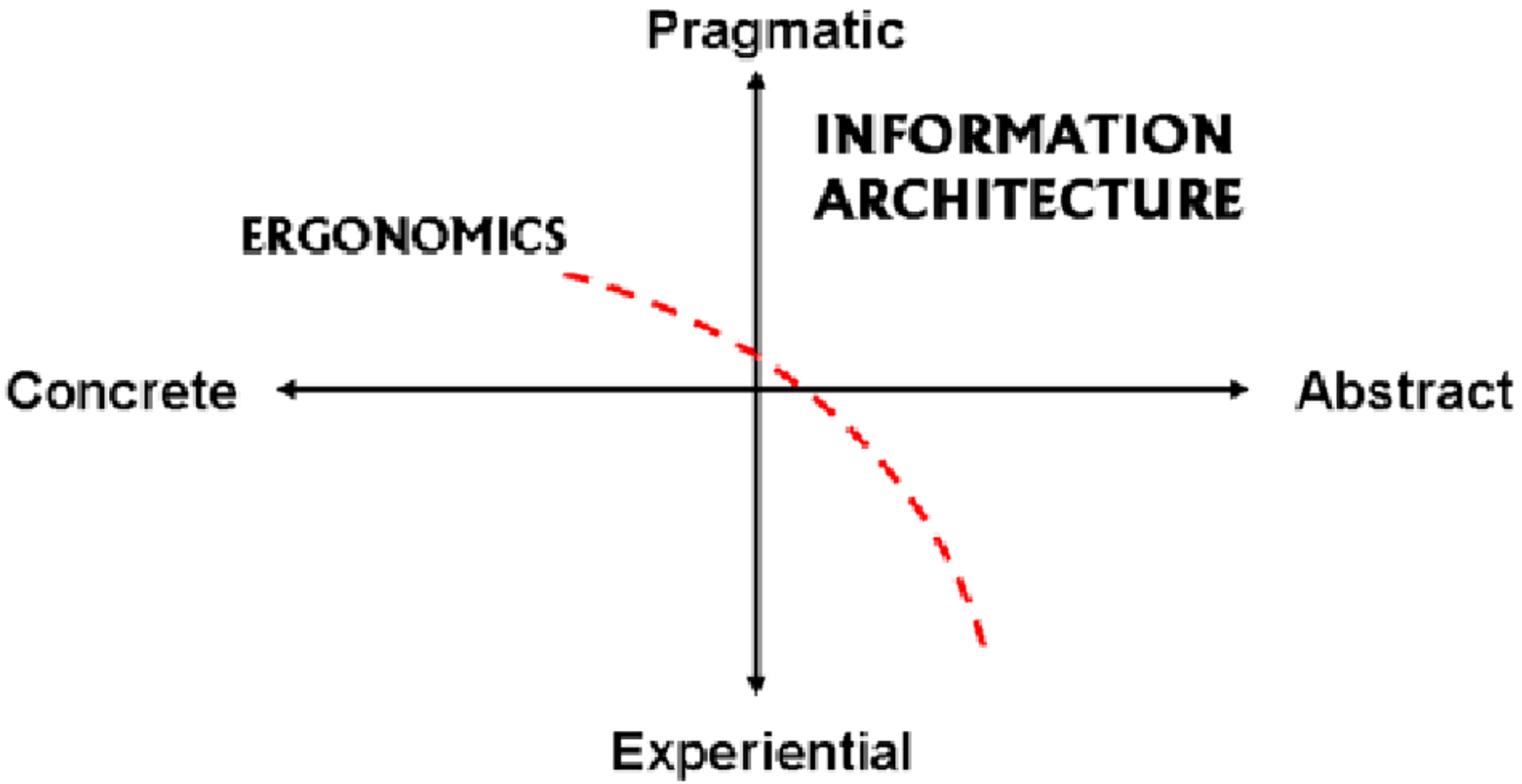
Or, if we only apply rigorous and deterministic methods and principles, what is the scope of applications we can apply them to?

==> A narrowly scoped Information Architecture assumes that applications exist to enforce some set of rules or constraints about information or processes

Designs with this scope and based on these methods and principles will presumably be more robust, reusable, and scalable

But will these methods and principles apply to enough contexts to be worth learning?

A Narrowly-Scoped IA



Methodology for a Narrowly-Scoped IA [1]

Create information and process models that capture requirements and "business rules" in a technology-neutral and robust way

Generate the application (or a "scaffold" for it) from the models

- The model can be directly used to generate the software
- The model can be interpreted by a generic software platform to configure its behavior
- Some combination of code generation and platform configuration may be employed

Methodology for a Narrowly-Scoped IA [2]

Evaluate the model-based application with usability or other *-ility techniques

Revise the models as suggested by these evaluations

Regenerate the application from the revised models

Model-Based User Interfaces

User interface design started as a distinct activity in the 1980s, and has been dominated by iterative and heuristic techniques ever since

In the 1990s the goal of model-based UIs emerged with the hope that automatic generation of window and menu layouts from information already present in the application data model can relieve the application designer of unnecessary work while providing an opportunity to automatically apply style rules to the interface design

Some people starting calling this the search for the Big Red Button, and in many cases it involved user interface modeling languages (expressed in XML) from which UI code would be generated

The model can be used to generate one or more application / UI "templates" and also guide the adaptation of "single source" content to make appropriate use of the interaction capabilities of the device or context in which it operates

XML and User Interfaces

Custom views of information for different users, devices, or context can be created by rendering the same XML document with different transforms and style sheets

Even if the complete UI cannot be generated in a completely automated way, models can generate prototypes, enabling more of the design space to be examined

XML Vocabularies for Describing User Interfaces

Many XML vocabularies for describing user interfaces have been developed; they differ in numerous ways but what may ultimately matter the most is whether Microsoft (or the "anyone but Microsoft") camps support them

- XUL - The elements of the XUL vocabulary include standard user interface components like menus, input controls, dialogs and tree controls, and keyboard shortcuts. Used by the Mozilla browser rendering engine called Gecko
- XAML - similar approach by Microsoft
- MXML - in Macromedia / Adobe Flash

Unfortunately, these XML vocabularies describe UIs at the presentation layer, not at the information model layer, so they fall short of the vision of MBUI

E-Forms

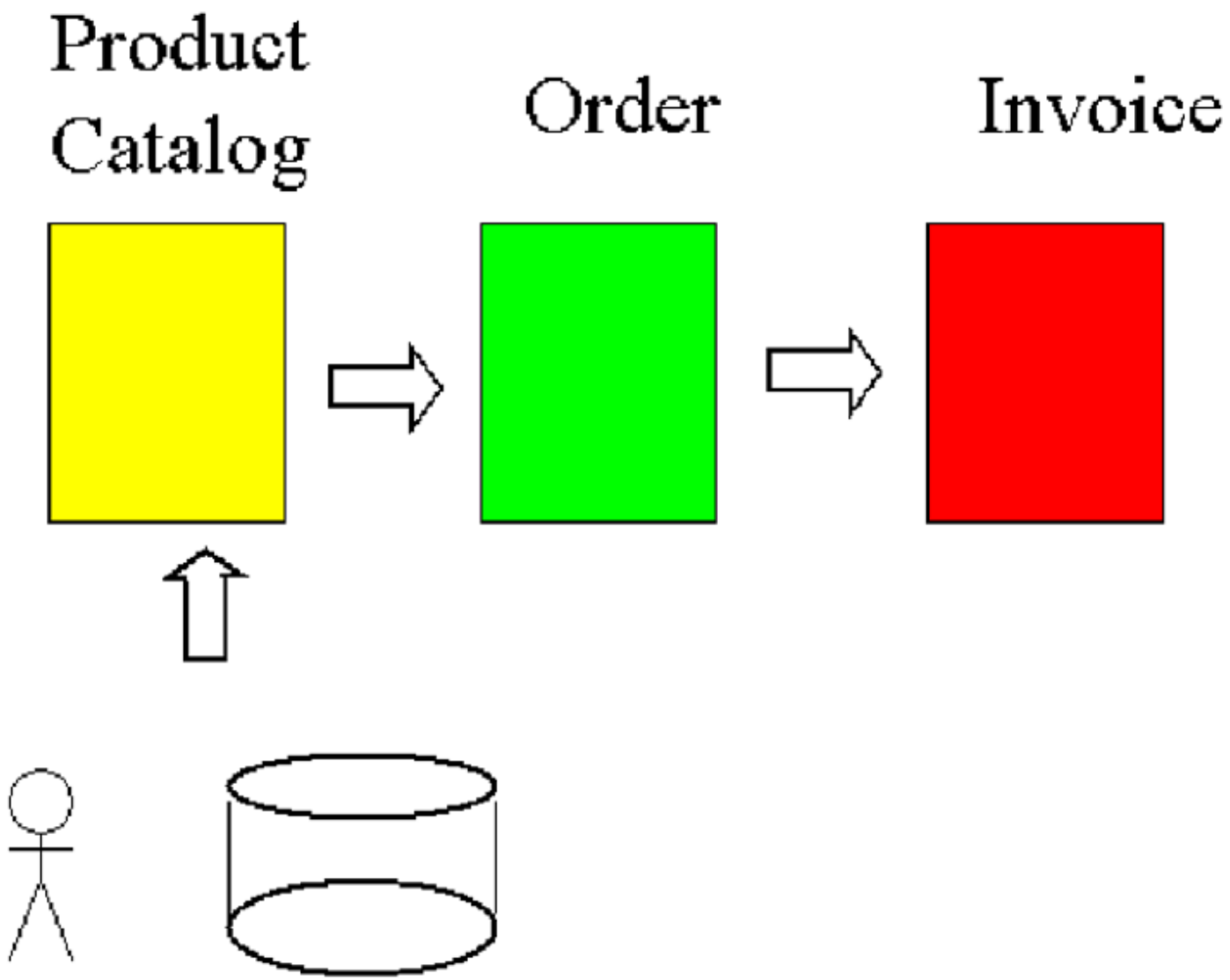
True model-based UI approaches are most promising for E-Form applications (especially those using XForms, a W3C specification)

Countless applications and services use a "fill-in-the-web-form" paradigm to automate processes that previously relied on printed forms

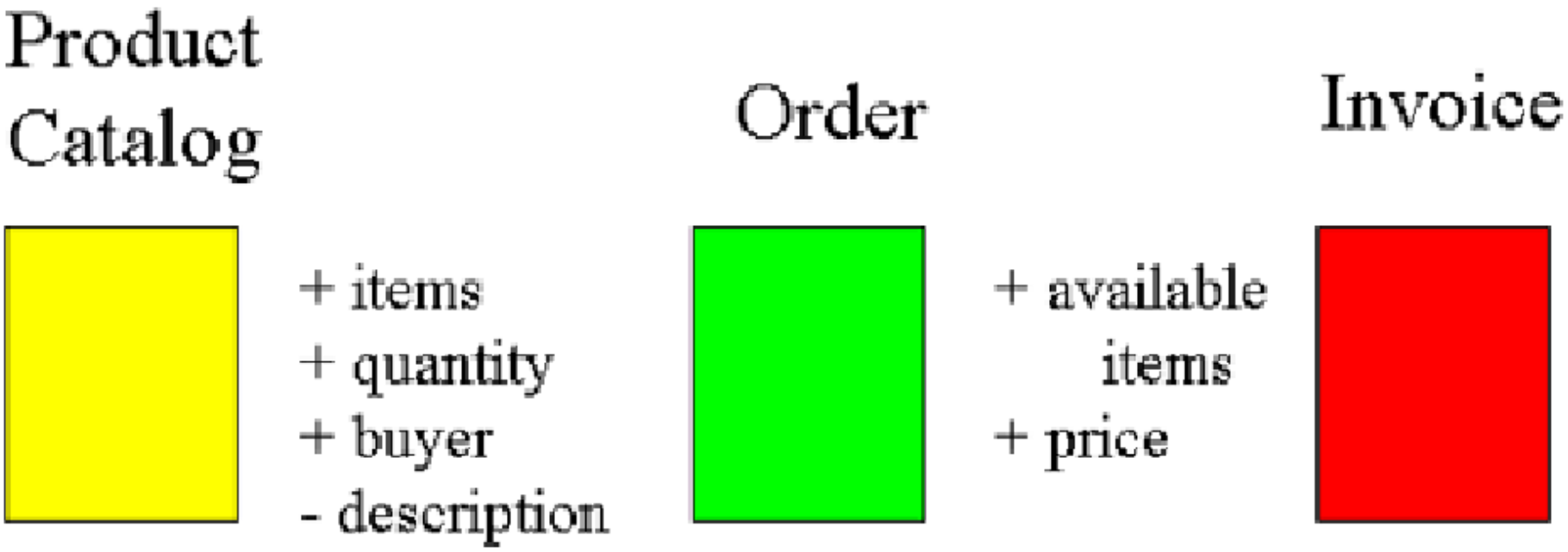
Filling out a form is creating a valid instance of the document type, and often the application is little more than "Webifying" a document interface to a legacy printed or client-server document application

For all but the simplest forms, however, complexity arises in the mapping of the logical model to the set of screens needed to collect the instance

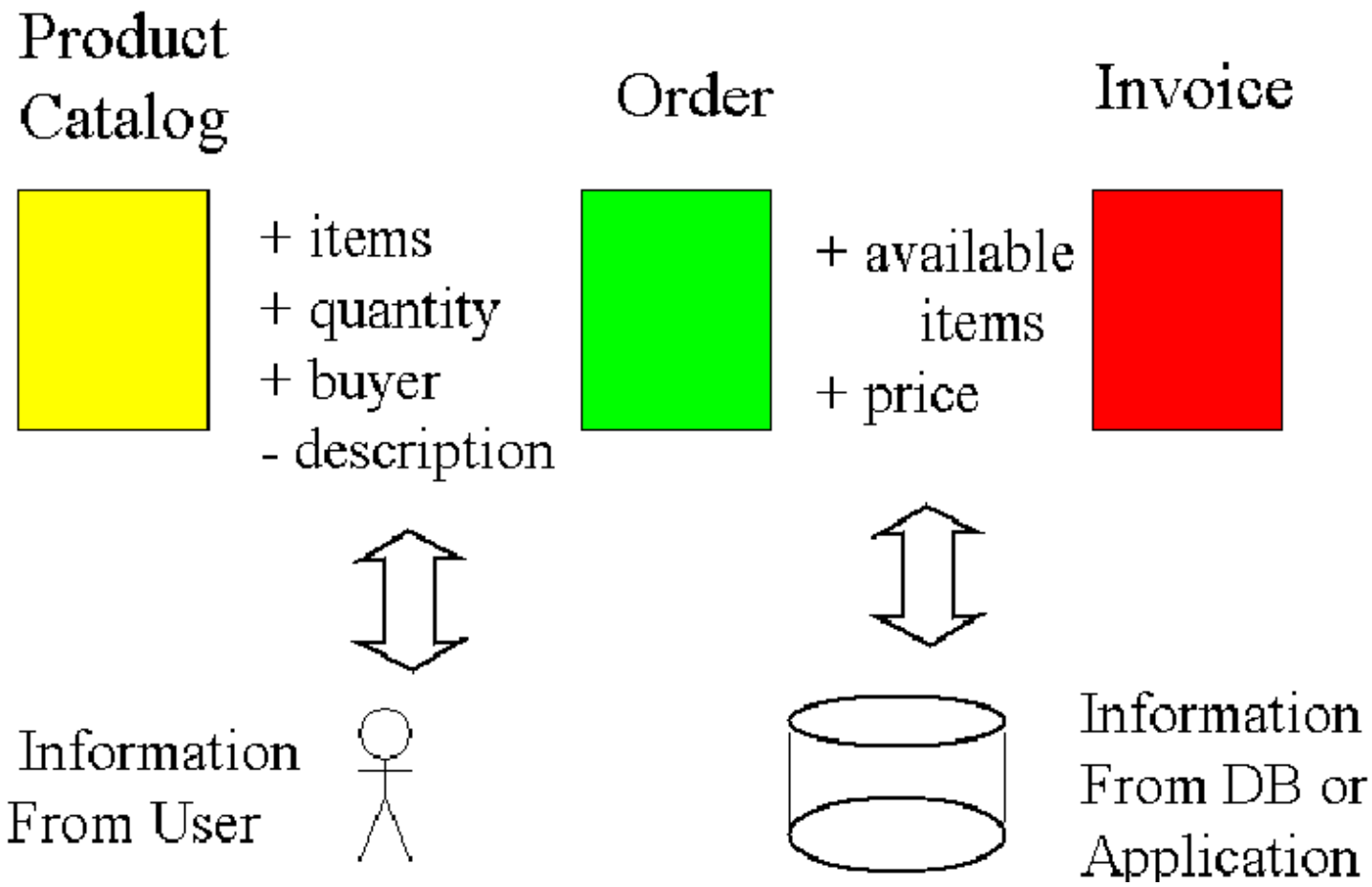
Forms and Workflow Applications



Model Components Reused in Transactions



Component-based User or Application Interfaces



Meeting in the Middle

The strongest proponents of MBUI are computer scientists who are comfortable with abstract models and techniques for code generation

Many MBUI proponents work in application contexts like mobile computing where the UI presentation repertoire is limited

Opponents of the MBUI approach argue that it de-emphasizes usability concerns and undermines the creative aspect of UI design

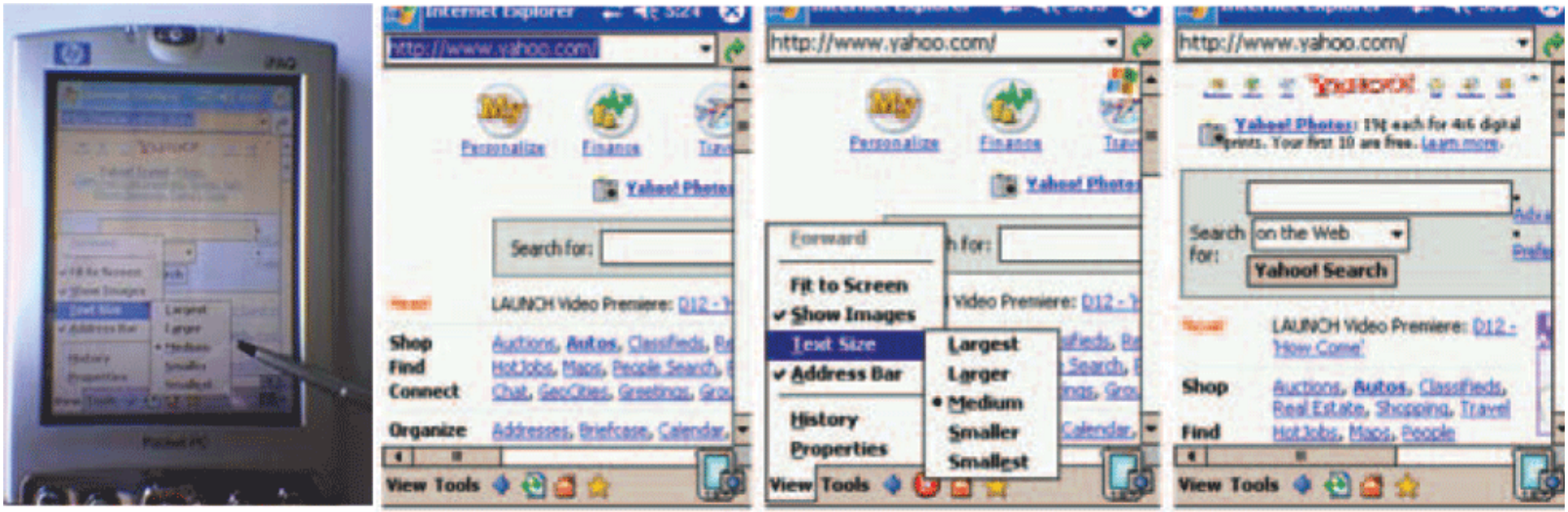
Many Small Red Buttons?

How can we "meet in the middle" to build UIs more efficiently with more predictable quality in UIs without eliminating creativity?

An alternative to the search for the BRB is the goal of partial automation for user interface generation:

- Tools that generate prototypes from specifications
- Tools that synthesize use cases into sequence diagrams
- Tools that merge sequence diagrams to hide states that have no UI implications
- Tools that generate UI skeletons or scaffolds while enforcing layout constraints
- Tools that generate a family of UIs via "graceful degradation" or "content adaptation"
- UI Design Patterns

UI and Content Adaptation - Example



Redefining "Information Architecture" [1]

Abstract patterns of information content or structure are sometimes called architectures

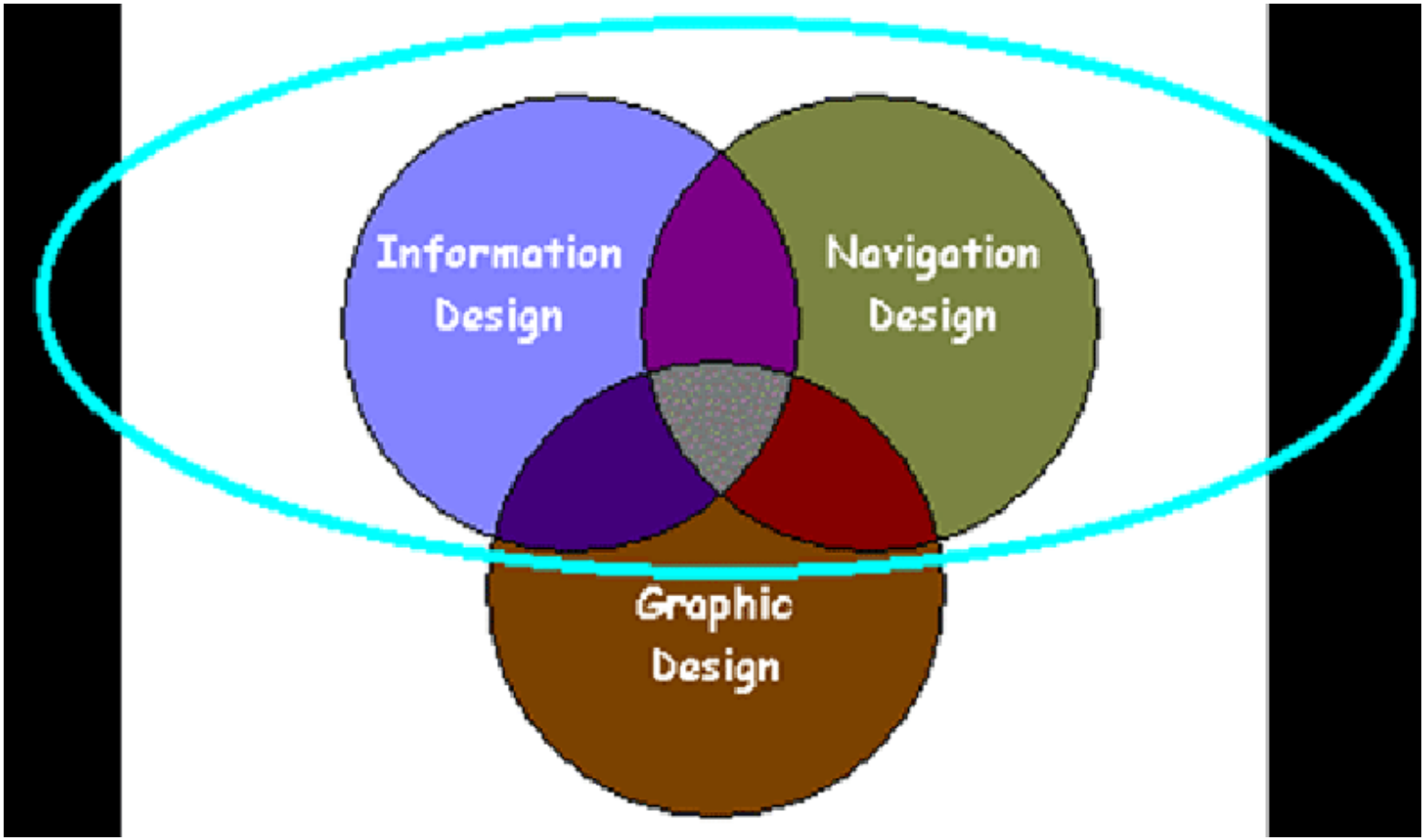
An architecture describes a system's components (or "building blocks") and their relationships with each other, using hierarchical and compositional structure to define the component boundaries

This gives an "architected" system an explicit model, in contrast with systems that are implemented incrementally without a master plan or without the effort to create reusable abstractions and components

So we define "Information Architecture" by combining the definitions of these concepts:

- "IA is designing an abstract and effective organization of information and then exposing that organization to facilitate navigation and information use"

Redefining Information Architecture [2]



A Semi-Rigorous Formula for Information Architecture

Information Architecture =
(((content + information structure) +
navigation structure) +
presentation structure) +
+ presentation design

The Most Important Principle for Information Architecture

From the 2nd lecture, "How to Think About Information"

- We say "the document is about … the photograph is about… the movie is about"
- We're expressing a distinction between information as conceptual or as content: and the physical container or medium, format, or technology in which the information is conveyed
- It is very useful to think abstractly about "information content" without making any assumptions or statements about the "presentation" or "rendition" or "implementation"

Separating content from its structure and presentation is the most important principle of Information Architecture

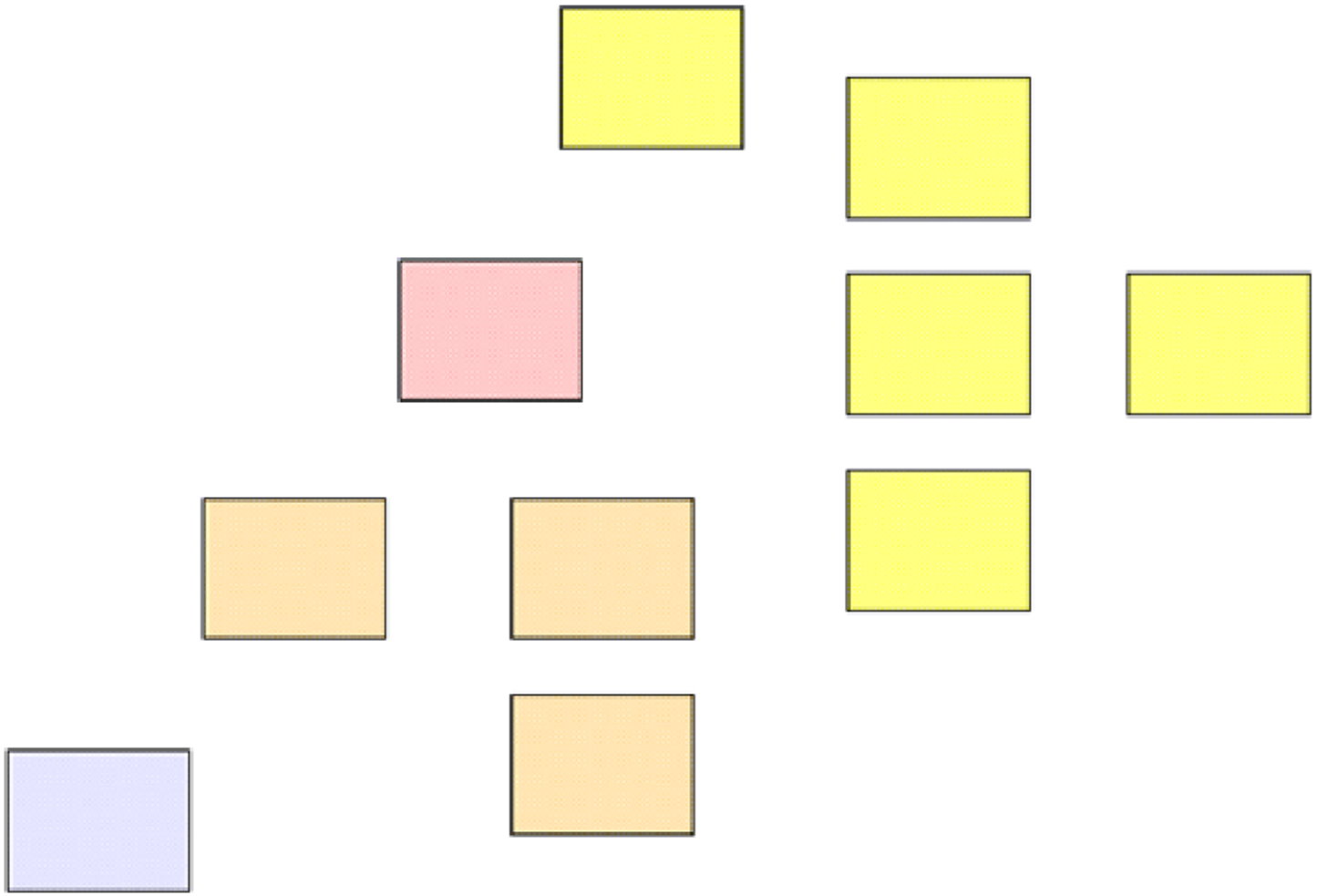
Three Types of "Stuff" or Kinds of Information

Content – "what does it mean" information

Structure
– "where is it" or "how it is organized or assembled" information

Presentation – "how does it look" or "how is it displayed" information

Content Components



Structural Information

Physical piece of a document or user interface (e.g. table, section, header, footer, panel, window)

Embodies the rules on how content components fit together, often hierarchical

Often driven by context of document use

Most applications and web sites are organized with a small set of structures:

- Lists/hierarchies
- Networks/links

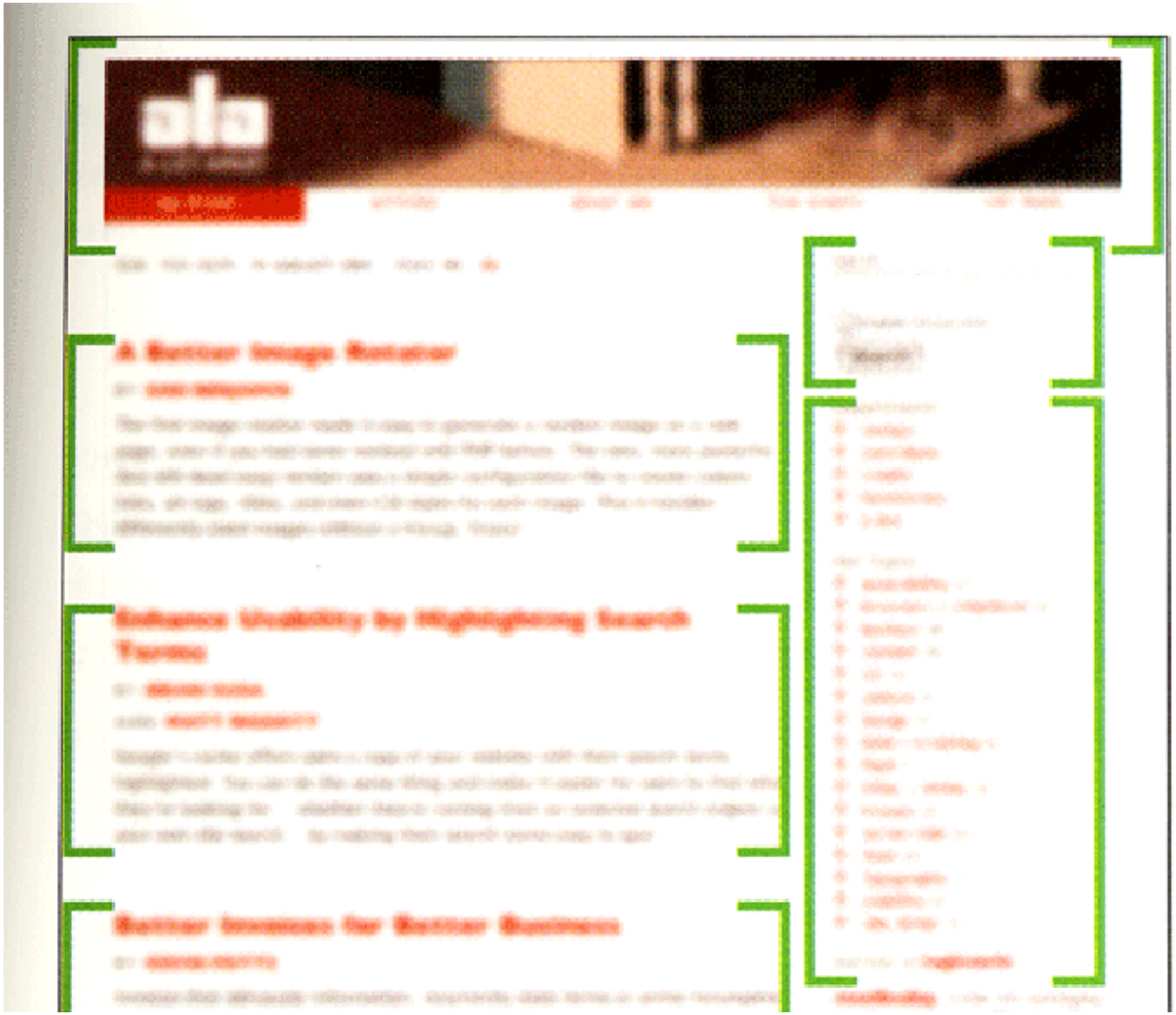
Applying Structure

The structural components provide the hierarchical "skeleton" or "scaffold" into which the content components are arranged; the structure remains fixed when the content changes

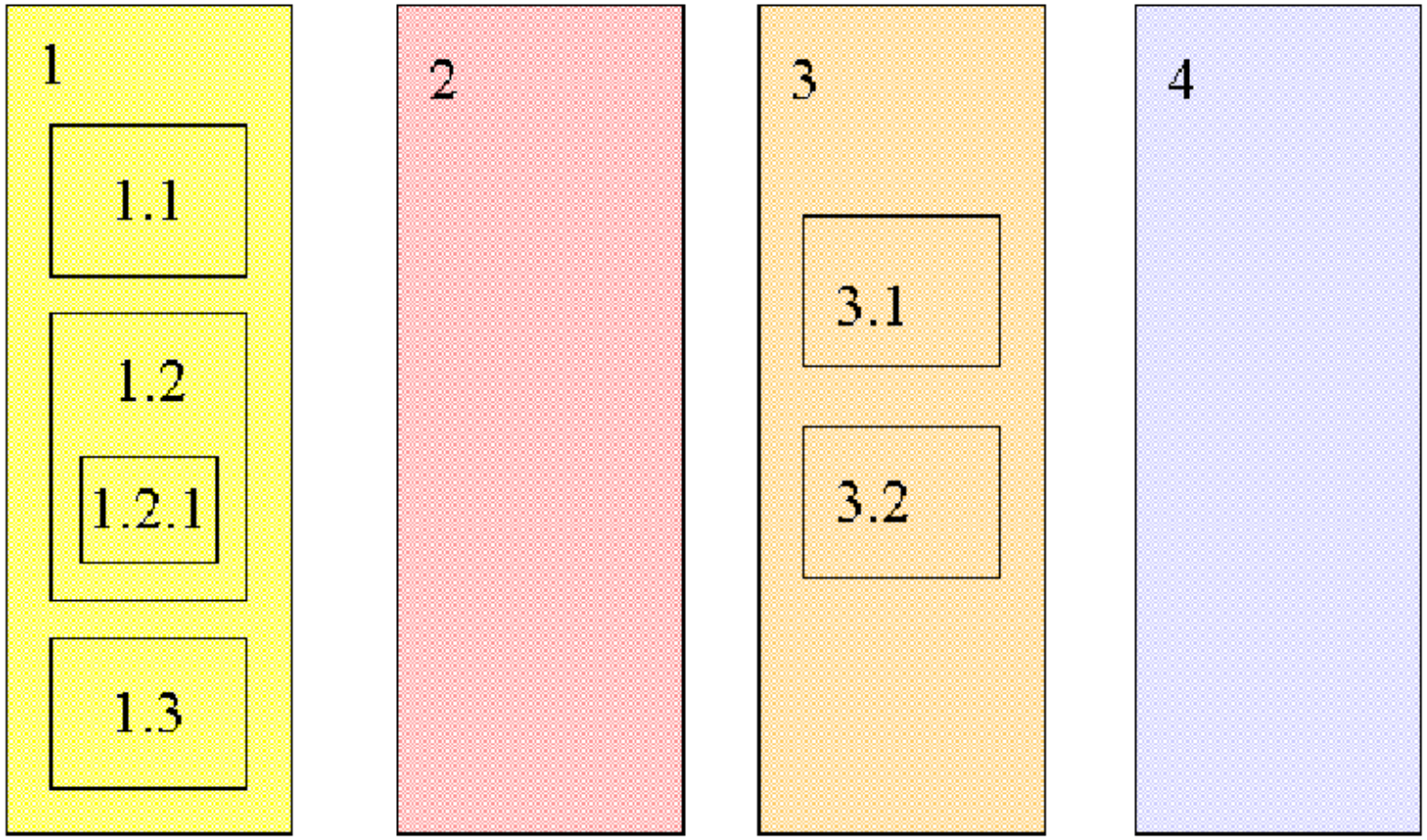
Structural components are often identified by the names attached to pieces of information – think of the outline or table of contents or lists of various kinds

Frequently a close relationship between structural and presentation items, especially in a paper document. This goes some way to explaining why the document-centric school places such strong emphasis on structural components.

Structure is Independent of Content



Structural Relationships Among Components Expressed as a Hierarchy



Lists

Common types of lists in user interfaces and applications:

- Lists of objects -- e.g., an inbox full of email messages
- Lists of actions or tasks -- e.g., browse, buy, sell, register
- Lists of subject categories (or facets) -- e.g, health, science, technology
- Lists of tools -- e.g., calendar, address book, notepad

Entry Points

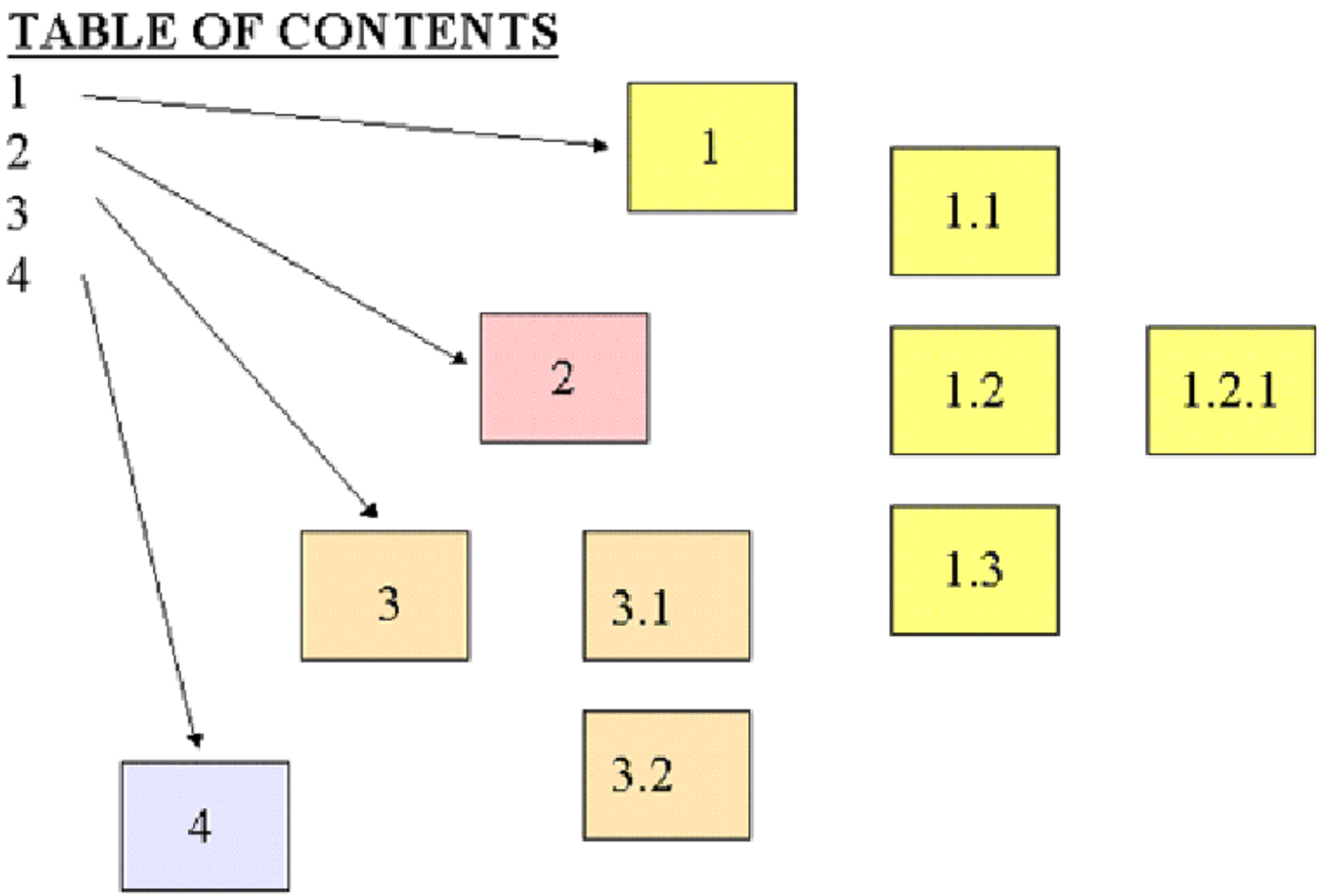
Similar to list structures are "entry points" -- structures that are "wrapped around" some set of content components to provide an organized way to access them

Most familiar examples are tables of contents and topical indexes; these are created from the names or other descriptive metadata for each component (which might first be extracted by processing the component content)

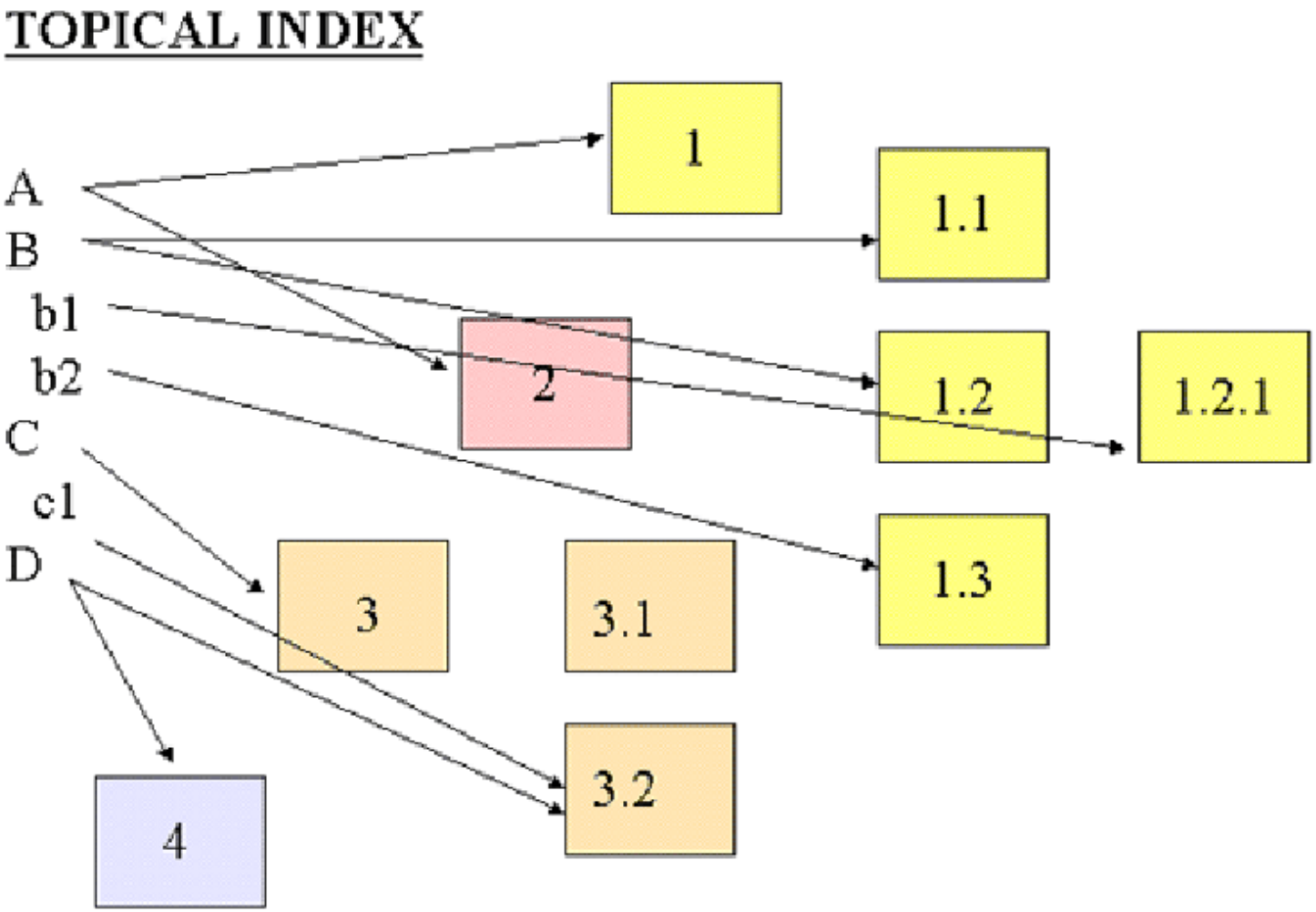
An entry point can be created as a static structure at design time, but preferably would be dynamically generated at run time

There are many similar examples of entry point structures generated from the names or descriptors of content components (Tables or Lists of content of type "X")

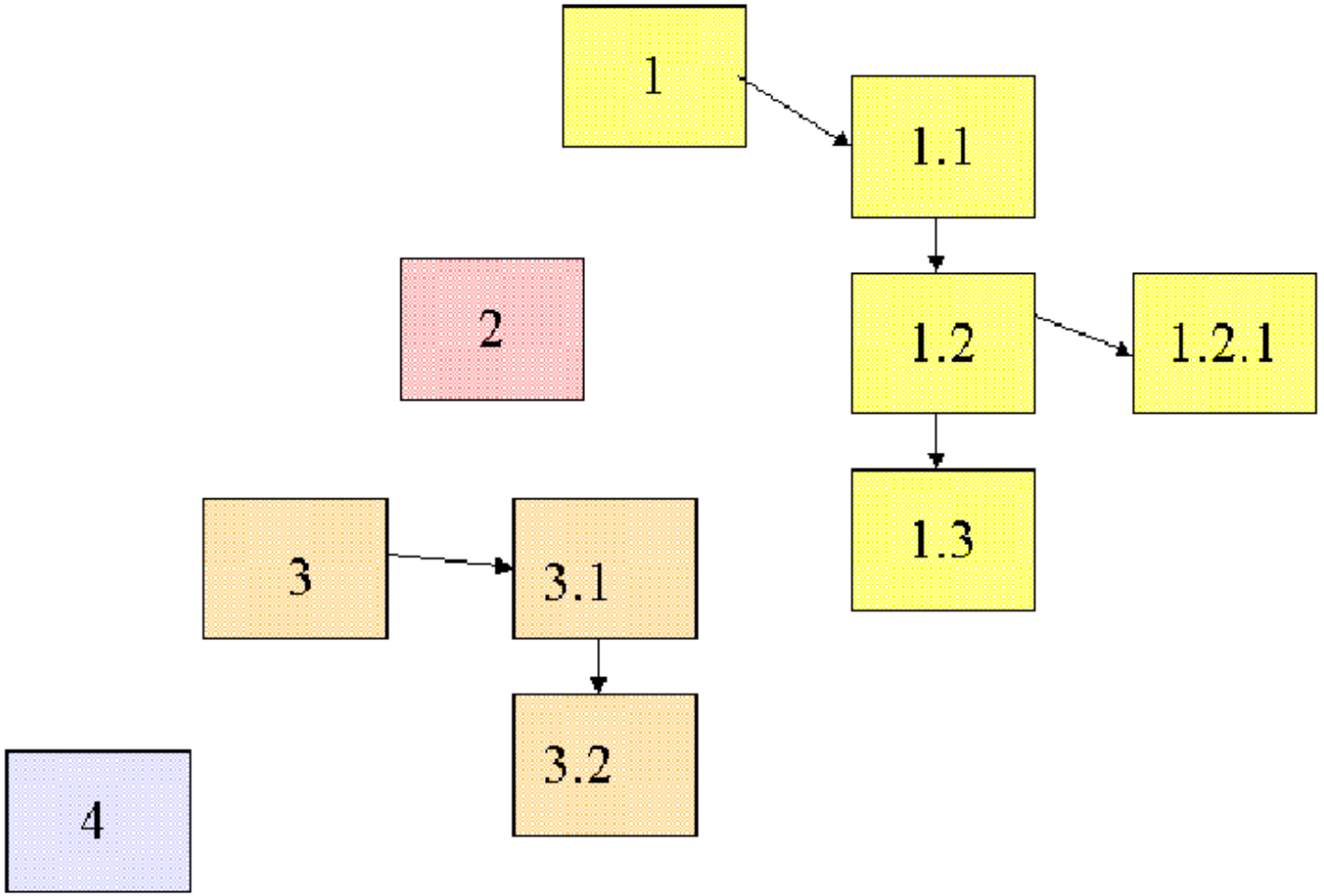
Table of Contents as "Entry Point"



Topical Index as "Entry Point"



Structural Relationships Among Components Expressed as a Network



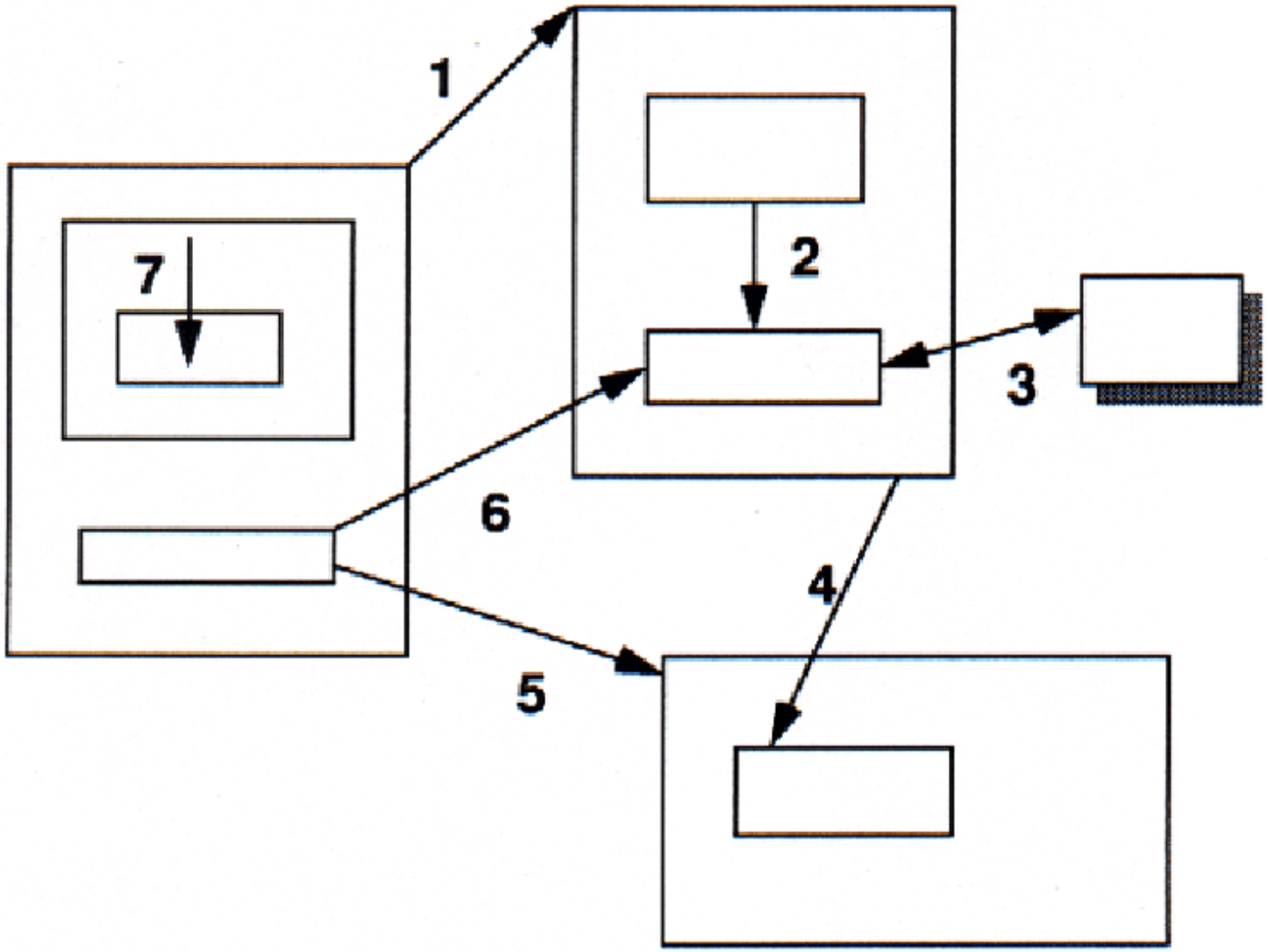
Links

Links are relationships between components that can express content as well as structural information

A link is represented in a logical model by its:

- Anchors -- the point, region, or span within the components to which it refers
- Type -- the semantics that the link relationship represents; not always explicit
- Directionality -- is the link one or two-way? Is the relationship meaningful in both directions? Does the reverse direction link mean the inverse?
- Cardinality -- 1 to 1 to many?

Link Structures



Navigation Structures

Navigation structures support finding or moving between components

- Forward or back in some structural organization
- Forward or back in a temporal organization (history list) or according to other relationships associated with the content components

Presentation Information

Human-oriented attributes for visual (or other sensory) differentiation (type font, type size, color, background, indentation, pitch, ...)

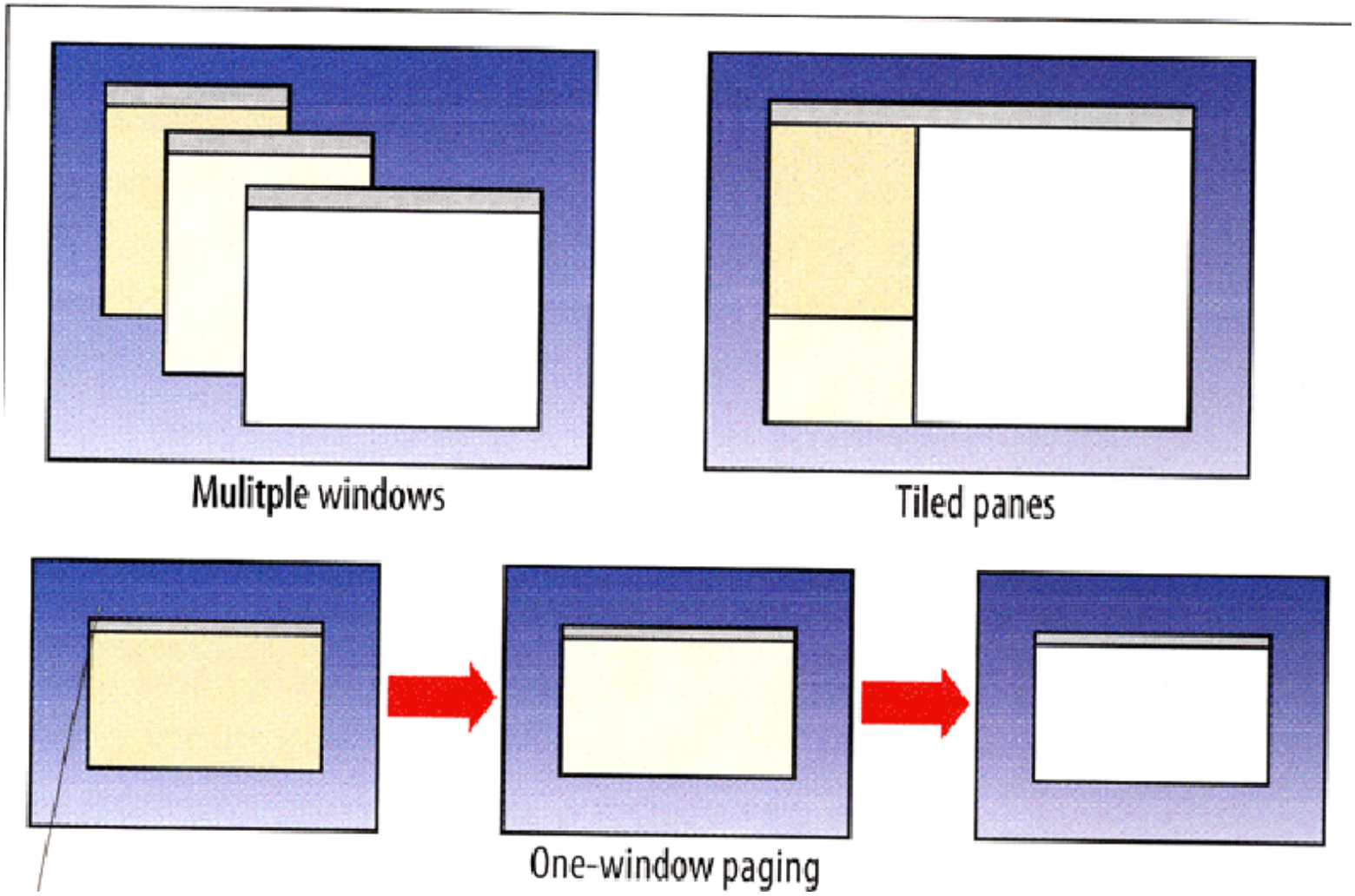
Good user interface design correlates this with structural or content information

Presentation affects structure and content by applying transformation rules to them

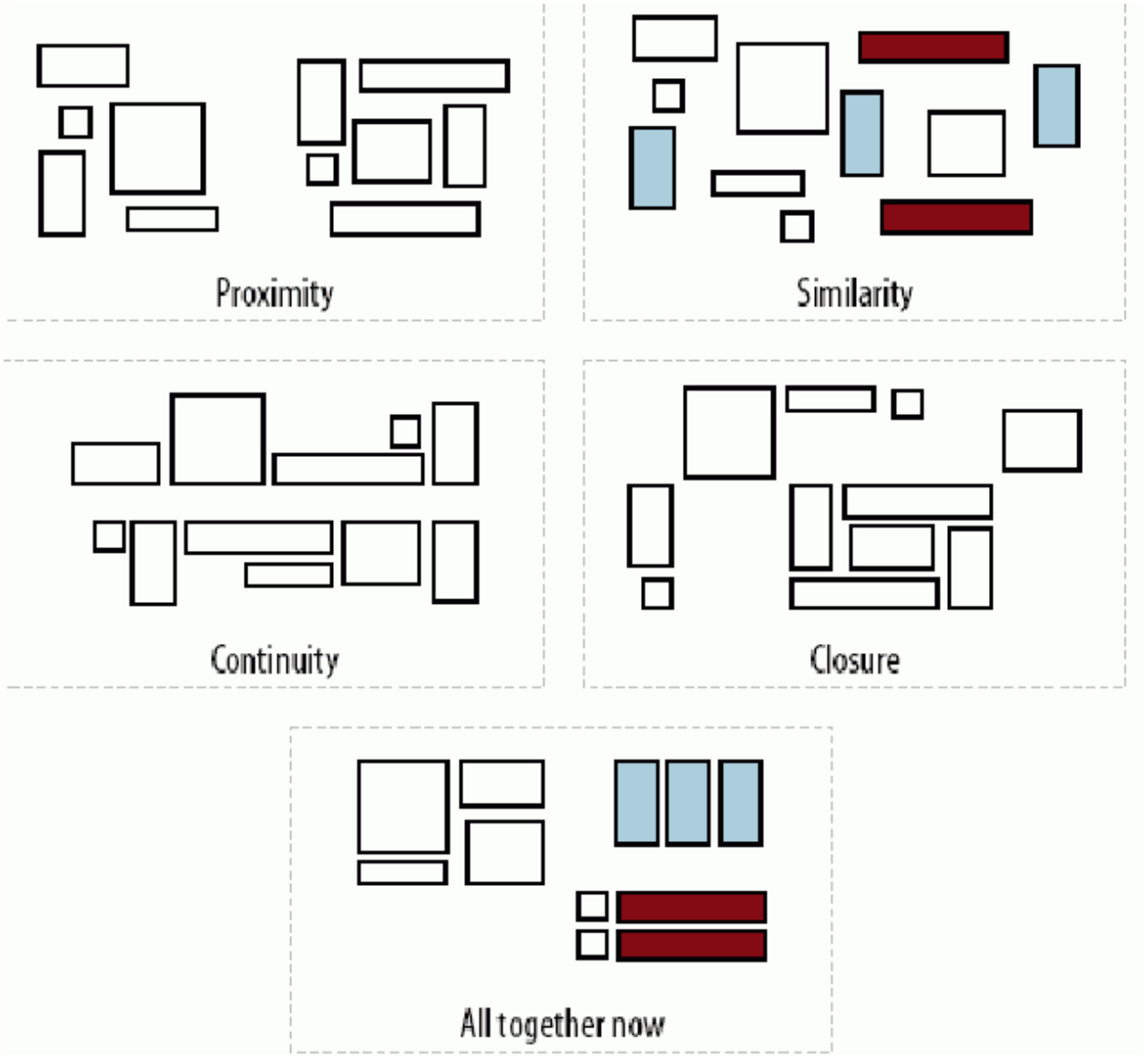
Some transform rules are explicit

Some transform rules are implicit or ambiguous or misleading

Binding Structure to Presentation - Alternatives



Gestalt Principles to Reinforce Structure



Internationalization

Internationalization is the design of products and services so that they can be easily localized for specific languages and cultures

Many of the architectural and design principles for internationalization are specializations of the "mother" principle to separate content from presentation:

- Leave text outside of graphics
- Separate strings/labels from code in scripts
- Use variable names that are not words in the scripts

Localization

Localization is the process of adapting the content and applications of a product or service to make it acceptable for a particular cultural or language market

Translation is the primary localization activity

The ease / automatability of localization depends on the principles and techniques used in internationalization

User Interface Design Patterns

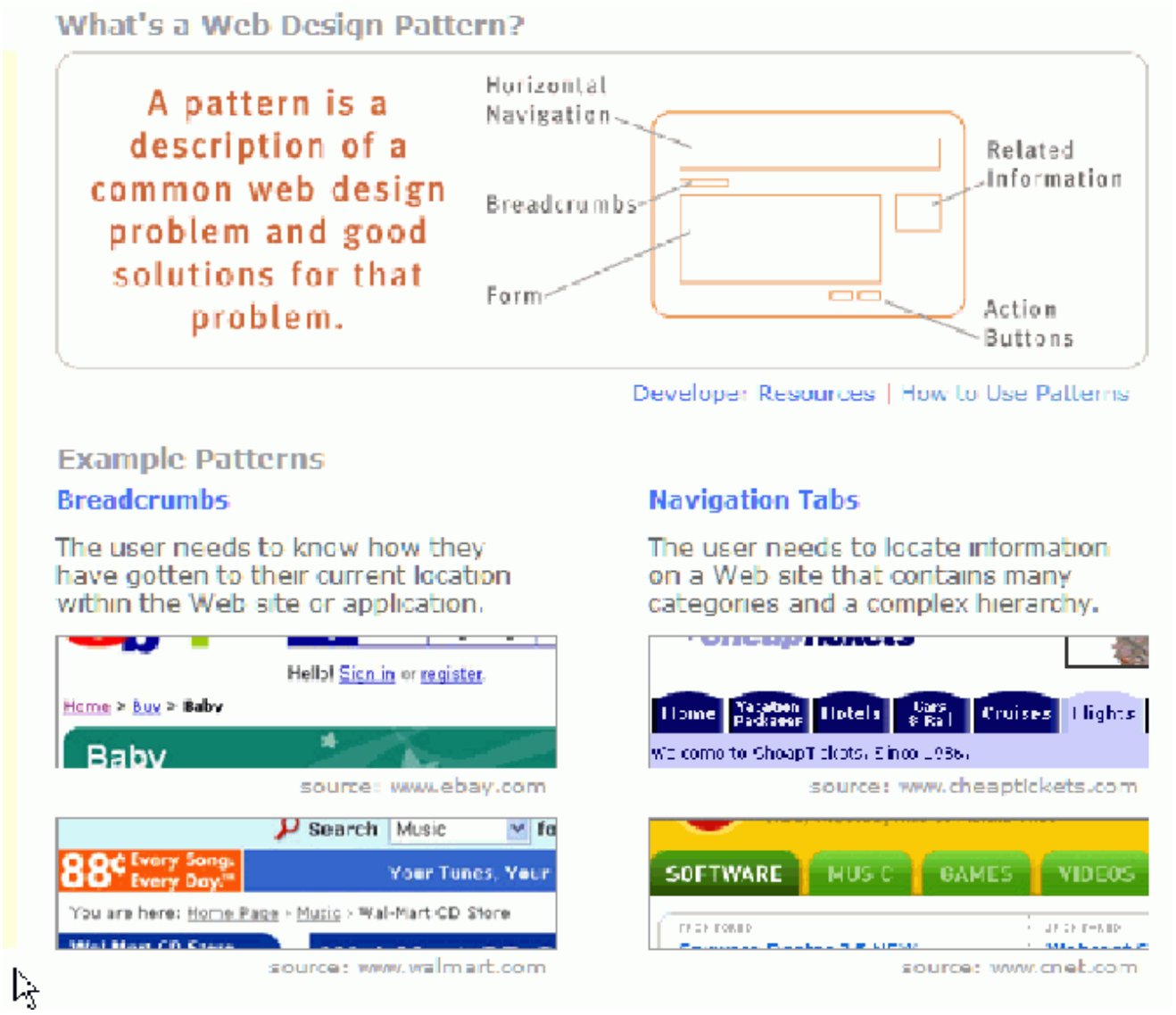
A very promising approach to making UI design more efficient, systematic and predictable

Patterns are solutions to common UI design problems that embody best practices for using structures and presentations for specific types of content

You can think of them as a very high level design vocabulary that guides designers toward better solutions than they would build if they started from less principled and tested starting points

Pattern libraries help even experienced designers, but provide the most value for relatively inexperienced ones

The UC Berkeley UIDP Project



The UC Berkeley UIDP Project

I-School '06: Kelly Snow, Mano Marks, Tim Dennis, David Hong

Target audience is UC Berkeley web developers, most of whom lack time or formal training in UI design

Very systematic and well-documented project to:

- Understand the skills and design practices of intended user community
- Analyze the kinds of applications they build
- Analyze existing pattern collections and "metamodels"
- Develop a methodology for developing patterns
- Develop 21 patterns in 5 categories (forms, information organization, navigation, profile management, and search)
- Design and test the user interface to the pattern repository
- Build and deploy the pattern repository

Readings for Lecture #20

Marti Hearst draft of Chapter 2, "Search User Interfaces" for 2nd edition of Modern Information Retrieval