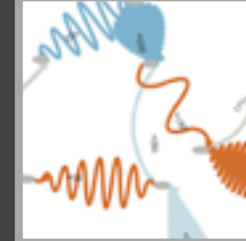
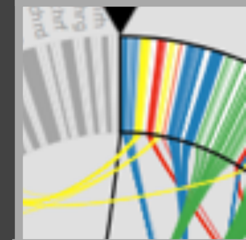
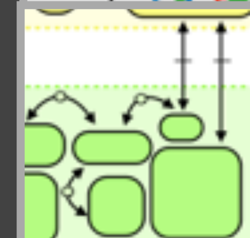
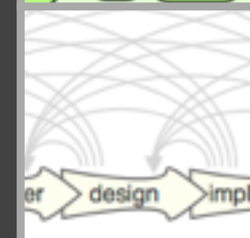
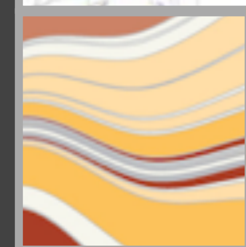
cs6630 | November 20 2014

VISUALIZATION MODELS

Miriah Meyer



administrivia . . .

- scalar data assignment due tonight
- transfer function assignment out

last time . . .

What is a vector field?

scalar field

$$s : \mathbb{E}^n \rightarrow \mathbb{R}$$

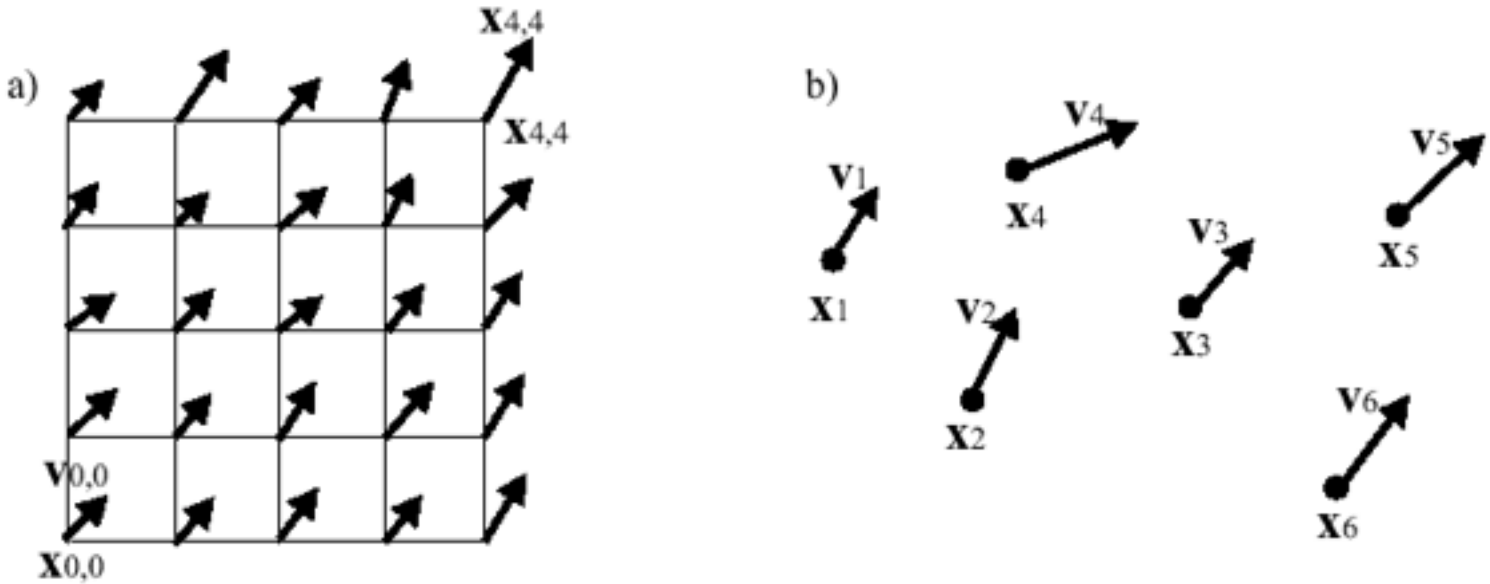
vector field

$$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$$

m will often be equal to n, but definitely not necessarily

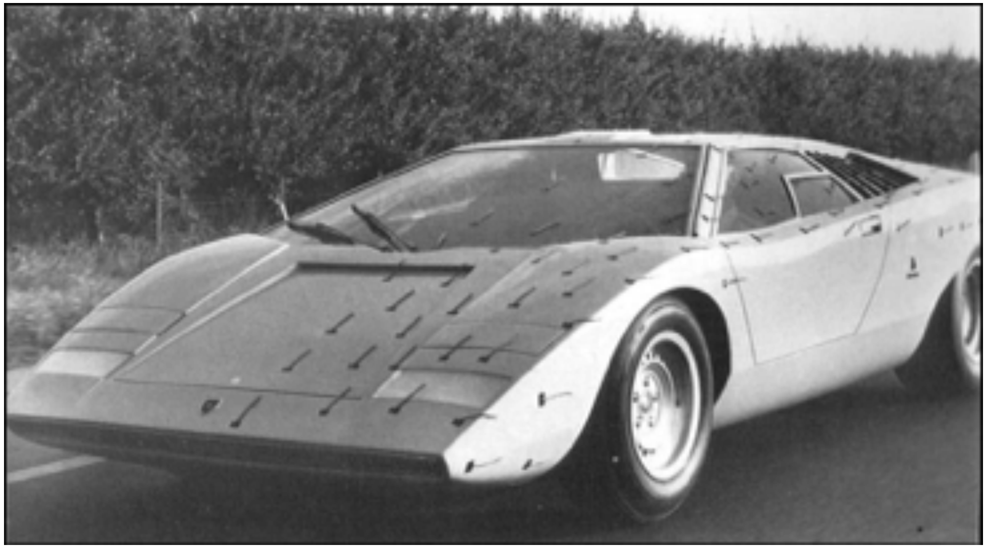
Flow Data

- Vector data on a 2D or 3D grid
- Additional scalar data may be defined per grid point
- Can either be on a regular grid (a) or scattered data points (b)



Smoke angel
A C-17 Globemaster III from the 14th Airlift Squadron, Charleston Air Force Base, S.C. flies off after releasing flares over the Atlantic Ocean near Charleston, S.C., during a training mission on Tuesday, May 16, 2006. The "smoke angel" is caused by the vortex from the engines.
(U.S. Air Force photo/Tech. Sgt. Russell E. Cooley IV)

http://de.wikipedia.org/wiki/Bild:Airplane_vortex_edit.jpg



http://autospeed.com/cms/A_108677/article.html

http://autospeed.com/cms/A_108677/article.html

Wool Tufts

scalar field

$s : \mathbb{E}^n \rightarrow \mathbb{R}$

vector field

$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$

tensor field

$\mathbf{T} : \mathbb{E}^n \rightarrow \mathbb{R}^{m \times b}$

scalar field

$$s : \mathbb{E}^n \rightarrow \mathbb{R}$$

$$s(\mathbf{x})$$

with $\mathbf{x} \in \mathbb{E}^n$

vector field

$$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{pmatrix}$$

with $\mathbf{x} \in \mathbb{E}^n$

tensor field

$$\mathbf{T} : \mathbb{E}^n \rightarrow \mathbb{R}^{m \times b}$$

$$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} c_{11}(\mathbf{x}) & \dots & c_{1b}(\mathbf{x}) \\ \vdots & & \vdots \\ c_{m1}(\mathbf{x}) & \dots & c_{mb}(\mathbf{x}) \end{pmatrix}$$

with $\mathbf{x} \in \mathbb{E}^n$

scalar field	vector field	tensor field
$s : \mathbb{E}^n \rightarrow \mathbb{R}$	$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$	$\mathbf{T} : \mathbb{E}^n \rightarrow \mathbb{R}^{m \times b}$
$s(\mathbf{x})$ with $\mathbf{x} \in \mathbb{E}^n$	$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{pmatrix}$ with $\mathbf{x} \in \mathbb{E}^n$	$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} c_{11}(\mathbf{x}) & \dots & c_{1b}(\mathbf{x}) \\ \vdots & & \vdots \\ c_{m1}(\mathbf{x}) & \dots & c_{mb}(\mathbf{x}) \end{pmatrix}$ with $\mathbf{x} \in \mathbb{E}^n$
	$\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$ 2D vector field	

scalar field	vector field	tensor field
$s : \mathbb{E}^n \rightarrow \mathbb{R}$	$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$	$\mathbf{T} : \mathbb{E}^n \rightarrow \mathbb{R}^{m \times b}$
$s(\mathbf{x})$ with $\mathbf{x} \in \mathbb{E}^n$	$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{pmatrix}$ with $\mathbf{x} \in \mathbb{E}^n$	$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} c_{11}(\mathbf{x}) & \dots & c_{1b}(\mathbf{x}) \\ \vdots & & \vdots \\ c_{m1}(\mathbf{x}) & \dots & c_{mb}(\mathbf{x}) \end{pmatrix}$ with $\mathbf{x} \in \mathbb{E}^n$
Could be the gradient of a scalar field.	$\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$ 2D vector field	

vector field

$$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$$

vector field

parameter-independent

$$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$$

steady vector field

vector field

parameter-independent

$$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$$

steady vector field

one-parameter-dependent

$$\mathbf{v} : \mathbb{E}^{n+1} \rightarrow \mathbb{R}^m$$

$$\mathbf{v}(x, y, t) = \begin{pmatrix} u(x, y, t) \\ v(x, y, t) \end{pmatrix}$$

unsteady vector field

	vector field	
parameter-independent	$\mathbf{v} : \mathbb{E}^n \rightarrow \mathbb{R}^m$ $\mathbf{v}(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}$	steady vector field
one-parameter-dependent	$\mathbf{v} : \mathbb{E}^{n+1} \rightarrow \mathbb{R}^m$ $\mathbf{v}(x, y, t) = \begin{pmatrix} u(x, y, t) \\ v(x, y, t) \end{pmatrix}$	unsteady vector field
two-parameter-dependent	$\mathbf{v} : \mathbb{E}^{n+2} \rightarrow \mathbb{R}^m$ $\mathbf{v}(x, y, s, t) = \begin{pmatrix} u(x, y, s, t) \\ v(x, y, s, t) \end{pmatrix}$	

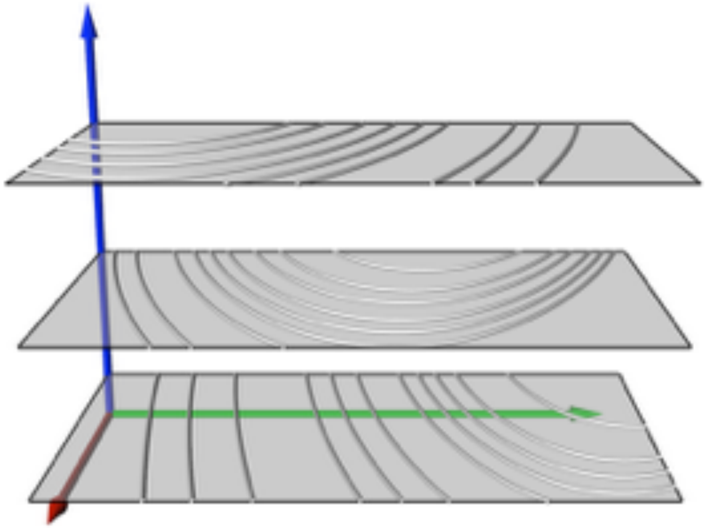
- **Divergence of v:**
- scalar field
- observe transport of a small ball around a point
  - expanding volume ➔ positive divergence
  - contracting volume ➔ negative divergence
  - constant volume ➔ zero divergence

$$\mathrm{div}\,\mathbf{v} = \frac{\delta u}{\delta x} + \frac{\delta v}{\delta y} + \frac{\delta w}{\delta z} = u_x + v_y + w_z$$

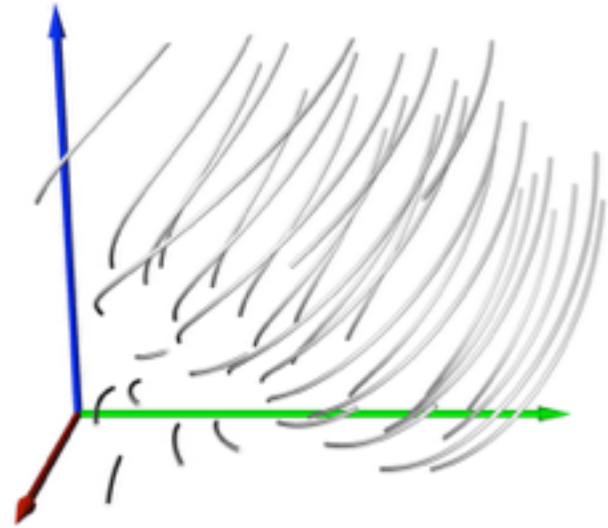
$\mathrm{div}\,\mathbf{v} \equiv 0 \Leftrightarrow$ $\mathbf{v}$ is incompressible

- **Curl of v:**
- vector field
- also called rotation (rot) or vorticity
- indication of how the field swirls at a point

$$\mathbf{curl\ v} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \dfrac{\delta}{\delta x} & \dfrac{\delta}{\delta y} & \dfrac{\delta}{\delta z} \\ u & v & w \end{vmatrix} = \begin{pmatrix} w_y - v_z \\ u_z - w_x \\ v_x - u_y \end{pmatrix}$$

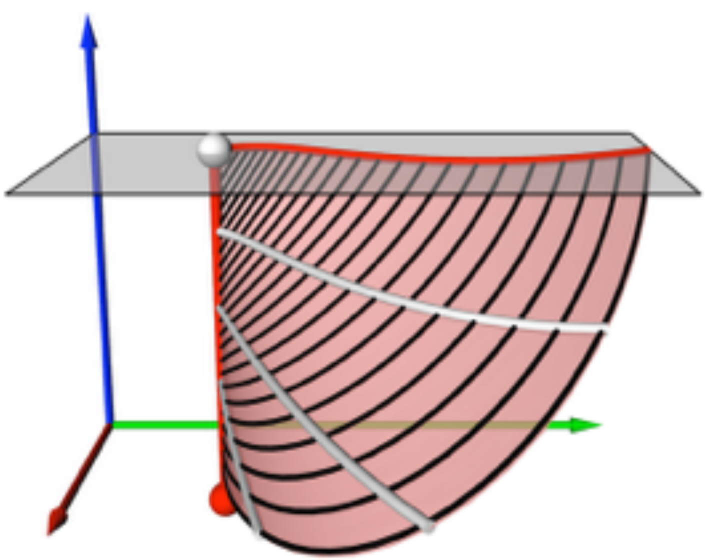


streamlines

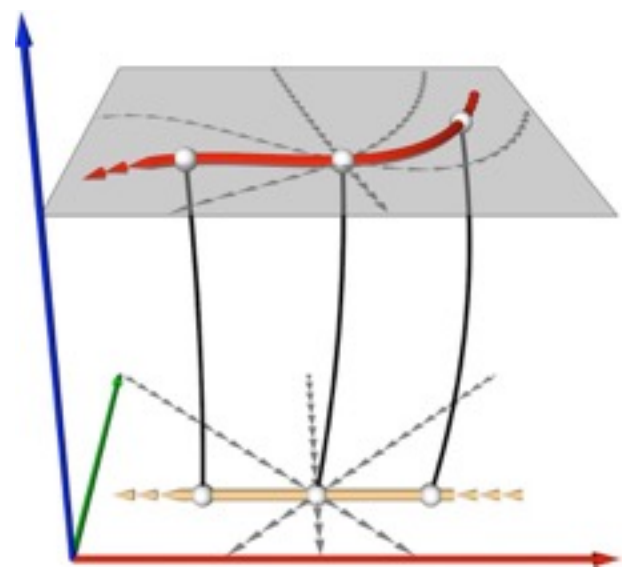


pathlines

streak lines



timelines



today . . .

- software architecture models
- design decision models
- process models

BUT FIRST...

visualization is a ***design process***

wicked problems

[Wicked Problems in Design Thinking, Buchanan 92]

wicked problems

- **alternative to linear, step-by-step approach to design**
  - approach: *problem definition* | *problem solution*
  - appealing as a "logical" understanding of design process

[Wicked Problems in Design Thinking, Buchanan 92]

wicked problems

- alternative to linear, step-by-step approach to design
    - approach: *problem definition | problem solution*
    - appealing as a "logical" understanding of design process
- Horst Rittel argued in the 1960s that most problems addressed by designers are "wicked"
    - *"class of social system problems which are ill formulated, where the information is confusing, where there are many clients and decision makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing"*

[Wicked Problems in Design Thinking, Buchanan 92]

wicked problems

- alternative to linear, step-by-step approach to design
  - approach: *problem definition | problem solution*
  - appealing as a "logical" understanding of design process
- Horst Rittel argued in the 1960s that most problems addressed by designers are "wicked"
  - *"class of social system problems which are ill formulated, where the information is confusing, where there are many clients and decision makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing"*
- determinacy versus indeterminacy
  - **linear model:** determinate problems have definite conditions
    - *designer should identify conditions and design solution*
  - **wicked model:** indeterminate problems have no definitive conditions or limits
    - *designer must discover or invent a particular subject out of the problem*

[Wicked Problems in Design Thinking, Buchanan 92]

10 properties of a wicked problem

(1) *Wicked problems* have no definitive formulation, but every formulation of a *wicked problem* corresponds to the formulation of a solution.
(2) *Wicked problems* have no stopping rules.
(3) Solutions to *wicked problems* cannot be true or false, only good or bad.
(4) In solving *wicked problems* there is no exhaustive list of admissible operations.
(5) For every *wicked problem* there is always more than one possible explanation, with explanations depending on the *Weltanschauung* of the designer.[39]
(6) Every *wicked problem* is a symptom of another, "higher level," problem.[40]
(7) No formulation and solution of a *wicked problem* has a definitive test.
(8) Solving a *wicked problem* is a "one shot" operation, with no room for trial and error.[41]
(9) Every *wicked problem* is unique.
(10) The *wicked problem* solver has no right to be wrong—they are fully responsible for their actions.

Richard Buchanan

Wicked Problems in Design Thinking

SUGGESTED READING

Introduction

This essay is based on a paper presented at "Colloque Recherches sur le Design: Incitations, Implications, Interactions," the first French university symposium on design research held October 1990 at l'Université de Technologie de Compiègne, Compiègne, France.

Despite efforts to discover the foundations of design thinking in the fine arts, the natural sciences, or most recently, the social sciences, design eludes reduction and remains a surprisingly flexible activity. No single definition of design, or branches of professionalized practice such as industrial or graphic design, adequately covers the diversity of ideas and methods gathered together under the label. Indeed, the variety of research reported in conference papers, journal articles, and books suggests that design continues to expand in its meanings and connections, revealing unexpected dimensions in practice as well as understanding. This follows the trend of design thinking in the twentieth century, for we have seen design grow from a *trade activity* to a *segmented profession* to a *field for technical research* and to what now should be recognized as a

- **software architecture models**
  - *focus on the structure of a software system in terms of its programmatic components*

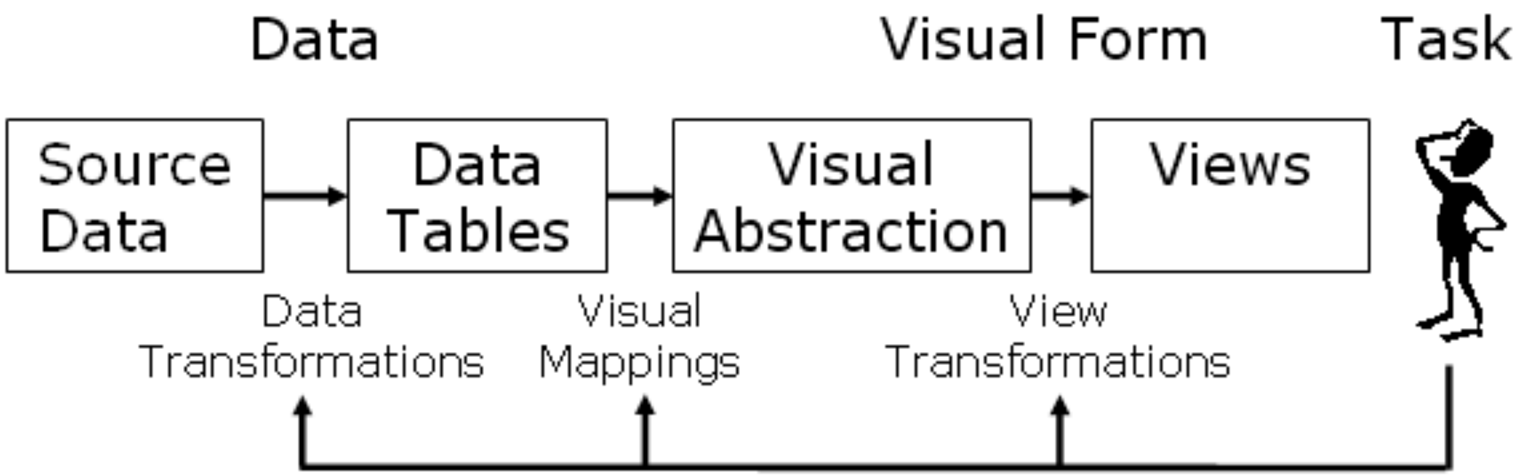
- **software architecture models**
    - *focus on the structure of a software system in terms of its programmatic components*
- **design decision models**
    - *describe and capture design decisions*

- **software architecture models**
  - *focus on the structure of a software system in terms of its programmatic components*
- **design decision models**
  - *describe and capture design decisions*
- **process models**
  - *describe stages with concrete actions a designer should engage in*

software architecture models

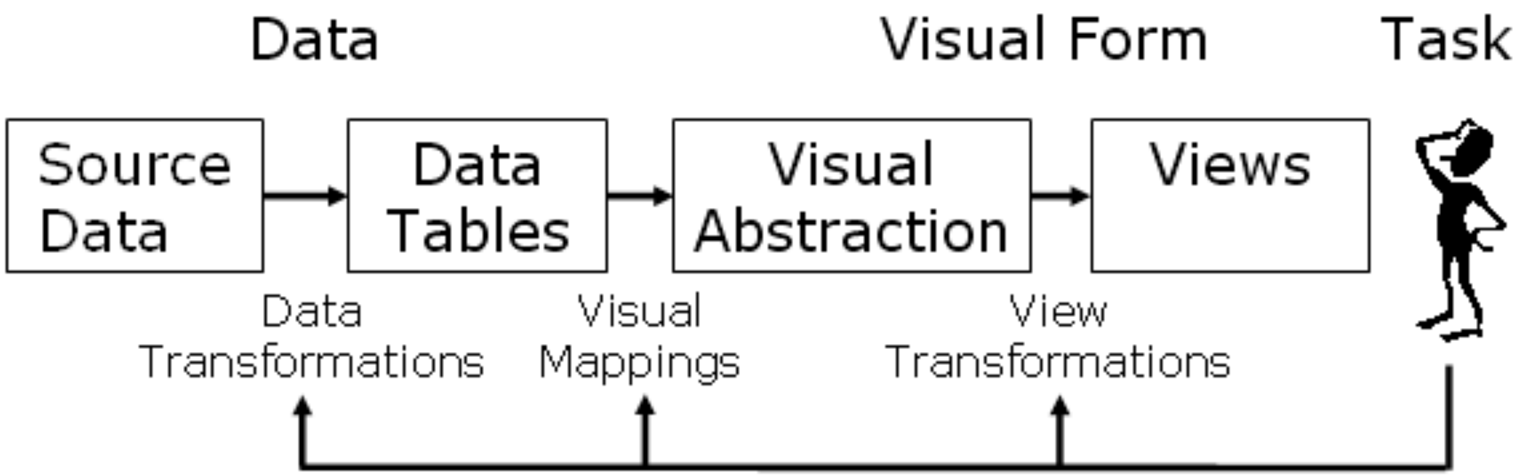
reference model

- software architecture pattern
    - *breaks up visualization (user) process into a series of discrete steps*



reference model

- **software architecture pattern**
  - *breaks up visualization (user) process into a series of discrete steps*



originally developed by Ed Chi as part of PhD dissertation, called the data state model; showed equivalence to data flow model used in existing toolkits like VTK

later interpreted by Card, Mackinlay, and Shneiderman, dubbing it the information visualization reference model

IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 12, NO. 5, SEPTEMBER/OCTOBER 2006

Software Design Patterns for Information Visualization

Jeffrey Heer and Maneesh Agrawala

Abstract—Despite a diversity of software architectures supporting information visualization, it is often difficult to identify, evaluate, and re-apply the design solutions implemented within such frameworks. One popular and effective approach for addressing such difficulties is to capture successful solutions in design patterns, abstract descriptions of interacting software components that can be customized to solve design problems within a particular context. Based upon a review of existing frameworks and our own experiences building visualization software, we present a series of design patterns for the domain of information visualization. We discuss the structure, context of use, and interrelations of patterns spanning data representation, graphics, and interaction. By representing design knowledge in a reusable form, these patterns can be used to facilitate software design, implementation, and evaluation, and improve developer education and communication.

Index Terms—Design patterns, information visualization, software engineering, object-oriented programming

1 Introduction

As recognition of the value of visualization has increased and the demand for visual analytics software has risen, visualization researchers have developed numerous software frameworks to meet these needs. By changing the cost structure governing the design and implementation of visualizations, such frameworks carry the potential to lower barriers to entry and increase the space of feasible visualization designs. Still, there is never a single tool or framework that is appropriate for all problems in a given domain. Developers often migrate between tools (*e.g.*, when developing on a new platform) or build their own systems (*e.g.*, to achieve functionality not available elsewhere). In either case, an understanding of the design solutions employed within existing tools could aid the programmer in learning and evaluating other frameworks and furthering their own development efforts. However, inspection of source code and design documents, if available, can prove difficult and tedious. Descriptions in the research literature often place more emphasis on novel features than on recurring design patterns. As a

Schmidt [18] has noted a number of benefits gained from incorporating design patterns into the development process. He found that patterns enabled widespread reuse of software architecture designs, improved communication within and across development teams, facilitated training of new programmers, and helped transcend ways of thinking imposed by individual programming languages. Schmidt also recommends that practitioners focus on developing patterns that are strategic to a domain of interest, while reusing general-purpose patterns (*e.g.*, those of [13]) as much as possible—an approach we now adopt for the design of information visualization software.

Previous research has applied the design pattern approach to visualization problems. Stolte et al. [21] introduce design patterns describing different forms of zooming within multi-scale visualizations. Chen [7] takes a more ambitious approach, suggesting high-level *visualization patterns* addressing general visualization concerns. He lists patterns such as Brushing, Linking, and Encoder,

- design patterns

- **design patterns**
  - *means of capturing time-tested design solutions and facilitating their reuse*

- design patterns
  - *means of capturing time-tested design solutions and facilitating their reuse*
- software design patterns

- **design patterns**
    - *means of capturing time-tested design solutions and facilitating their reuse*
- **software design patterns**
    - *descriptions of communicating objects and classes that are customized to solve design problems within a particular context*

- **design patterns**
  - *means of capturing time-tested design solutions and facilitating their reuse*
- **software design patterns**
  - *descriptions of communicating objects and classes that are customized to solve design problems within a particular context*
- **specific patterns for visualization**
  - *related to: application structure, data handling, graphics, and interaction*

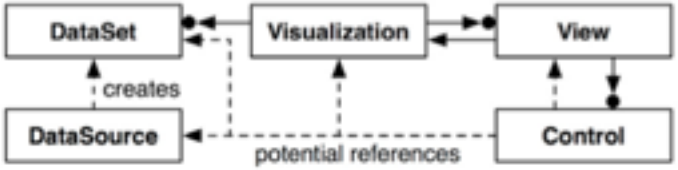
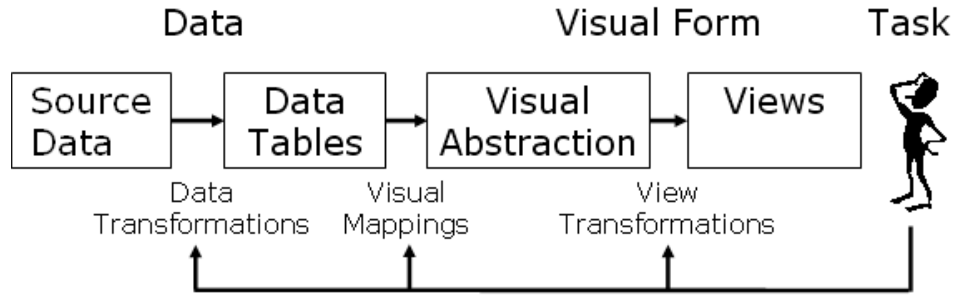
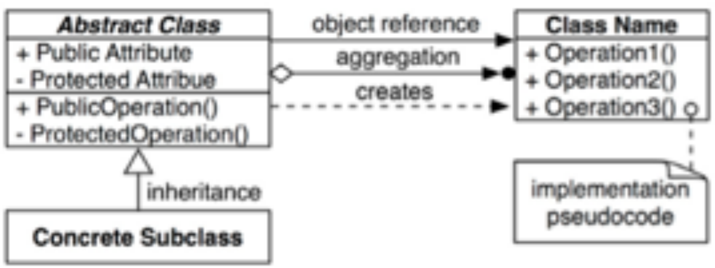


Figure 2. The Reference Model Pattern. A visualization manages visual models for one or more data sets, separating visual attributes (location, size, color, geometry, etc) from the abstract data. One or more views provide a graphical display of the visualization, while control modules process user input and may trigger updates at any level of the system.





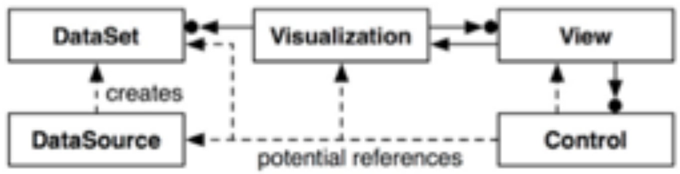
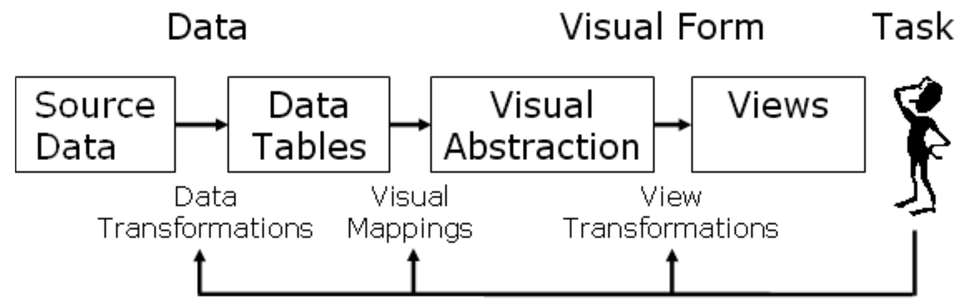


Figure 2. The Reference Model Pattern. A visualization manages visual models for one or more data sets, separating visual attributes (location, size, color, geometry, etc) from the abstract data. One or more views provide a graphical display of the visualization, while control modules process user input and may trigger updates at any level of the system.



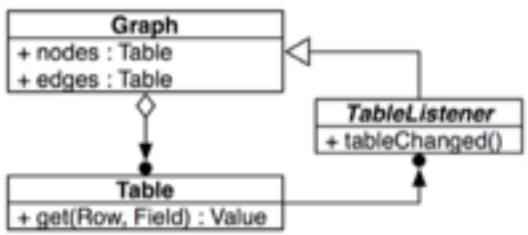


Figure 5. The Relational Graph Pattern. Network structures are implemented using relational data tables to represent node and edge data. Edge tables maintain foreign keys which reference incident nodes.

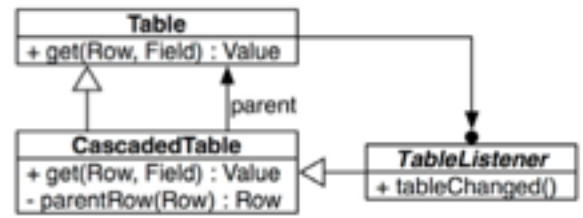


Figure 4. The Cascaded Table Pattern. A cascaded table inherits values from a parent table instance. The cascaded table may manage its own set of data columns, potentially shadowing columns in the parent. Column references not found in the child table are resolved against the parent table.

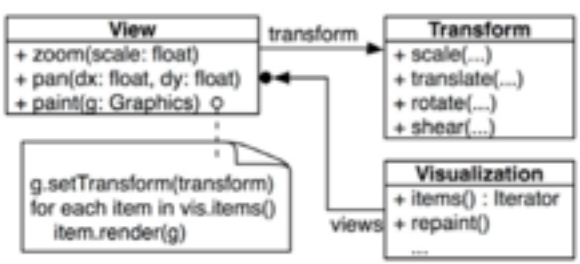


Figure 12. The Camera Pattern. A view component maintains an affine transformation matrix that is applied to visual items when rendering. The affine transform matrix can be used to specify translation, rotation, scale, and shearing transformations on the geometry of the view.

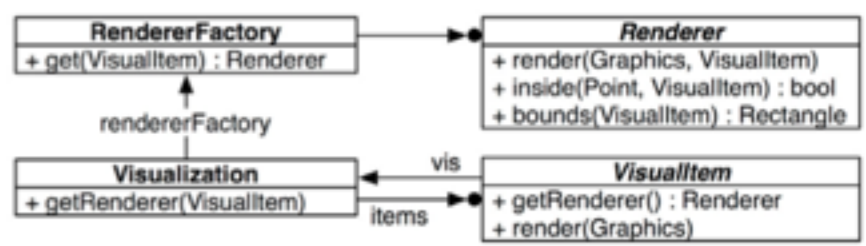
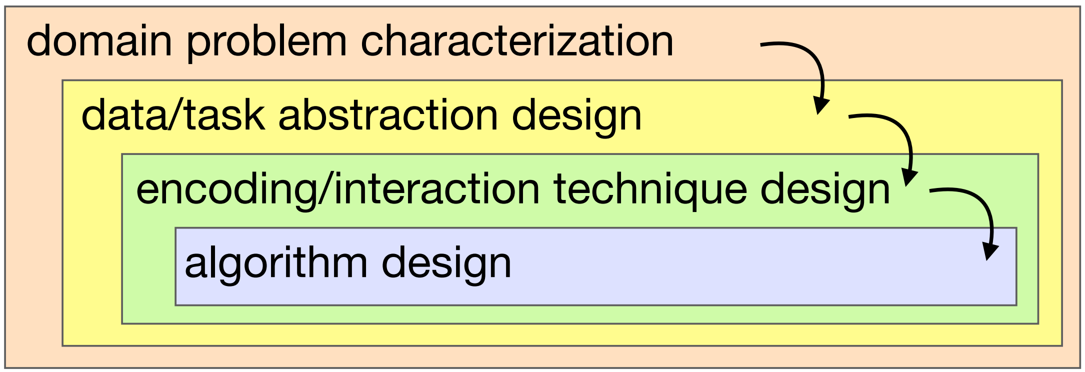


Figure 10. The Renderer Pattern. The mapping between items and their visual appearance is determined using Renderer modules, responsible for drawing, interior point testing, and bounds calculation. A RendererFactory can be used to assign Renderers to items based on current conditions, such as data attribute values or the zoom level.

design decision models

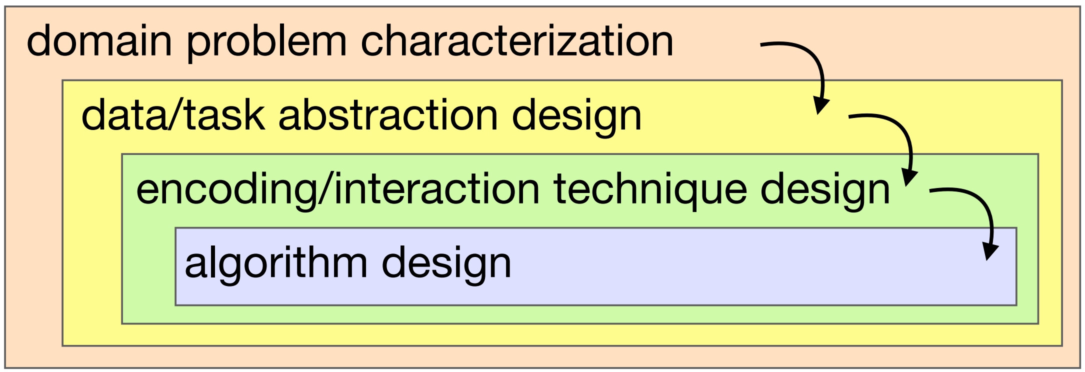
design decision models *vs* process models



nested model

design decision model: describes levels of design inherent to, and should be considered in, the creation of a tool

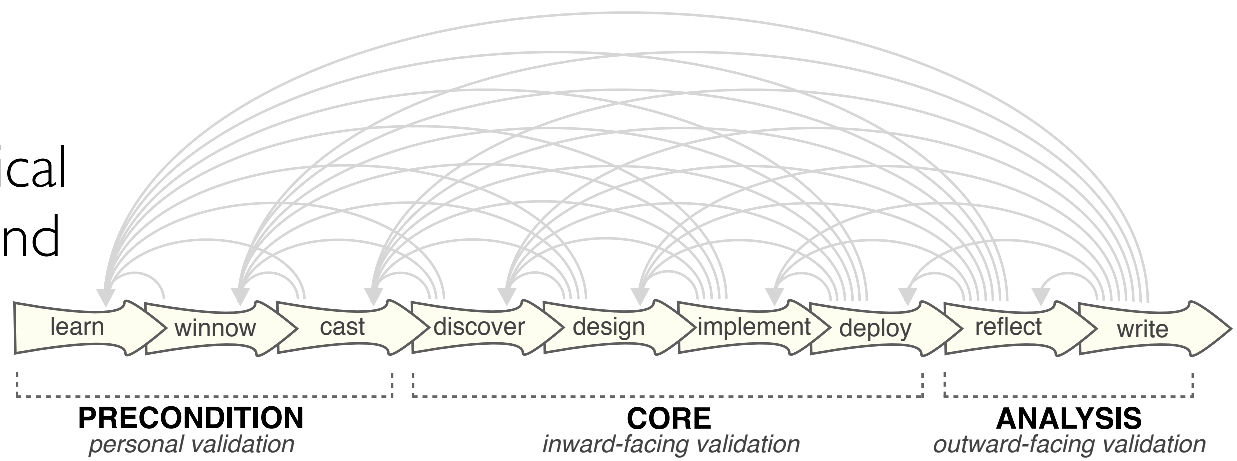
design decision models *vs* **process models**



nested model

design decision model: describes levels of design inherent to, and should be considered in, the creation of a tool

process model: gives practical advice in how to design and develop a tool



9-stage framework

A Nested Model for Visualization Design and Validation

Tamara Munzner
University of British Columbia
Department of Computer Science

How do you show your system is good?

- so many possible ways!
  - algorithm complexity analysis
  - field study with target user population
  - implementation performance (speed, memory)
  - informal usability study
  - laboratory user study
  - qualitative discussion of result pictures
  - quantitative metrics
  - requirements justification from task analysis
  - user anecdotes (insights found)
  - user community size (adoption)
  - visual encoding justification from theoretical principles

Contribution

- nested model unifying design and validation
  - guidance on when to use what validation method
  - different threats to validity at each level of model
- recommendations based on model

Four kinds of threats to validity

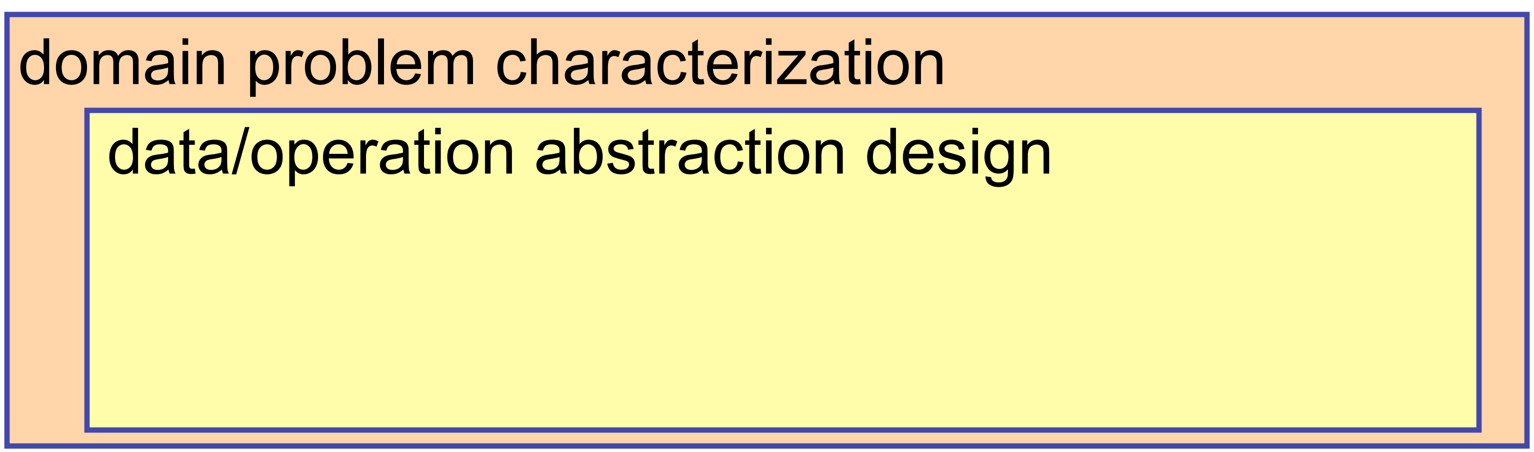
Four kinds of threats to validity

- wrong **problem**
  - they don't do that

domain problem characterization

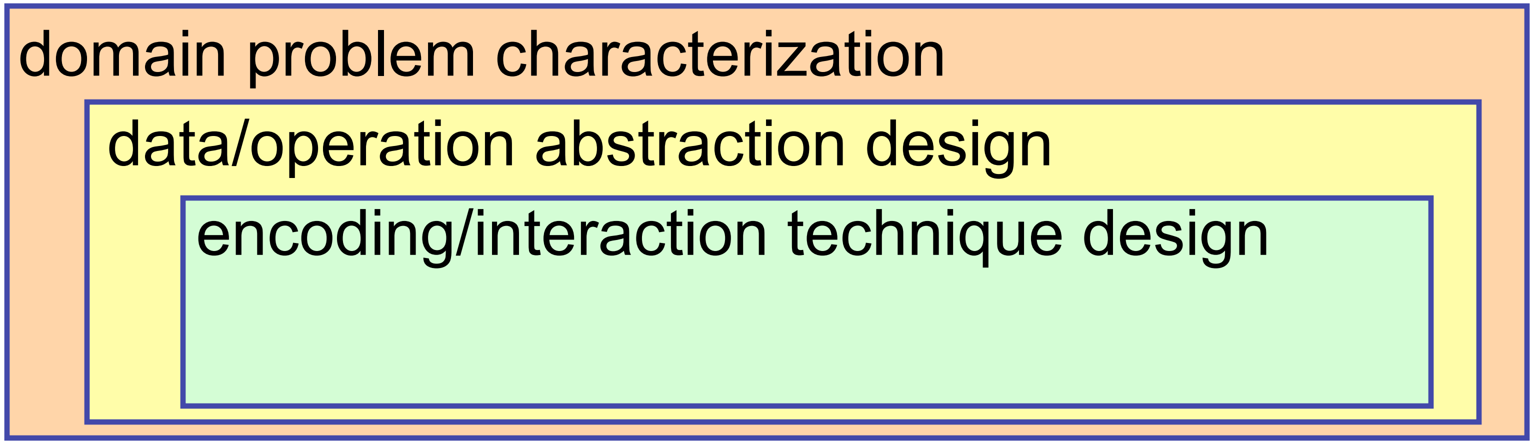
Four kinds of threats to validity

- wrong problem
  - they don’t do that
- wrong **abstraction**
  - you’re showing them the wrong thing



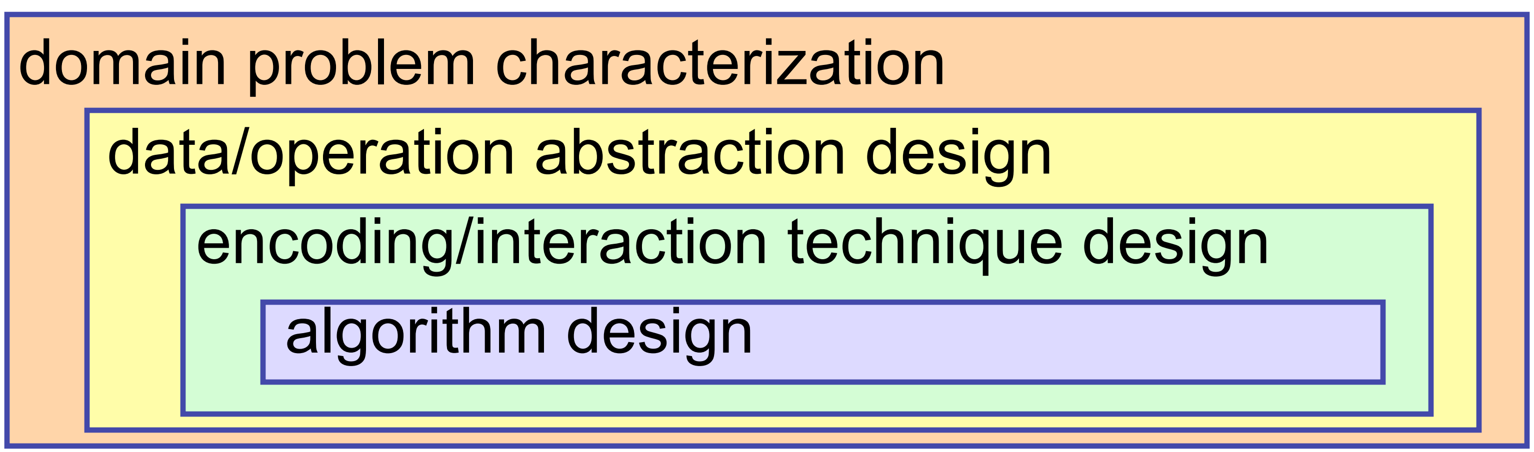
Four kinds of threats to validity

- wrong problem
  - they don’t do that
- wrong abstraction
  - you’re showing them the wrong thing
- wrong encoding/interaction **technique**
  - the way you show it doesn’t work



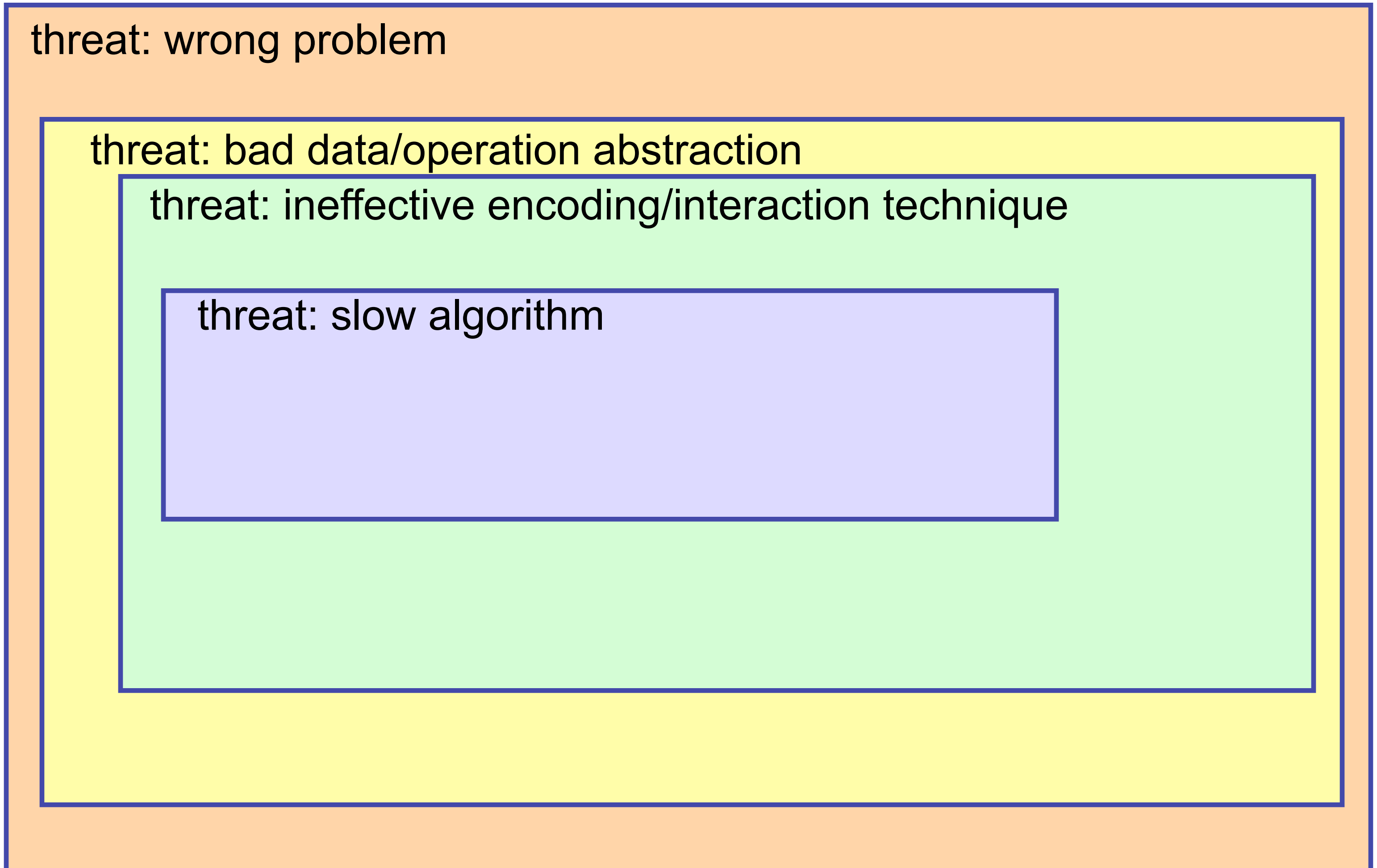
Four kinds of threats to validity

- wrong problem
  - they don’t do that
- wrong abstraction
  - you’re showing them the wrong thing
- wrong encoding/interaction technique
  - the way you show it doesn’t work
- wrong **algorithm**
  - your code is too slow



Match validation method to contributions

- each validation works for only one kind of threat to validity



Analysis examples

MatrixExplorer. Henry and Fekete. InfoVis 2006.

observe and interview target users
justify encoding/interaction design
measure system time/memory
qualitative result image analysis

LiveRAC. McLachlan, Munzner, Koutsofios, and North. CHI 2008.

observe and interview target users
justify encoding/interaction design
qualitative result image analysis
field study, document deployed usage

An energy model for visual graph clustering. (LinLog) Noack. Graph Drawing 2003

qualitative/quantitative image analysis

Effectiveness of animation in trend visualization. Robertson et al. InfoVis 2008.

lab study, measure time/errors for operation

Interactive visualization of genealogical graphs. McGuffin and Balakrishnan. InfoVis 2005.

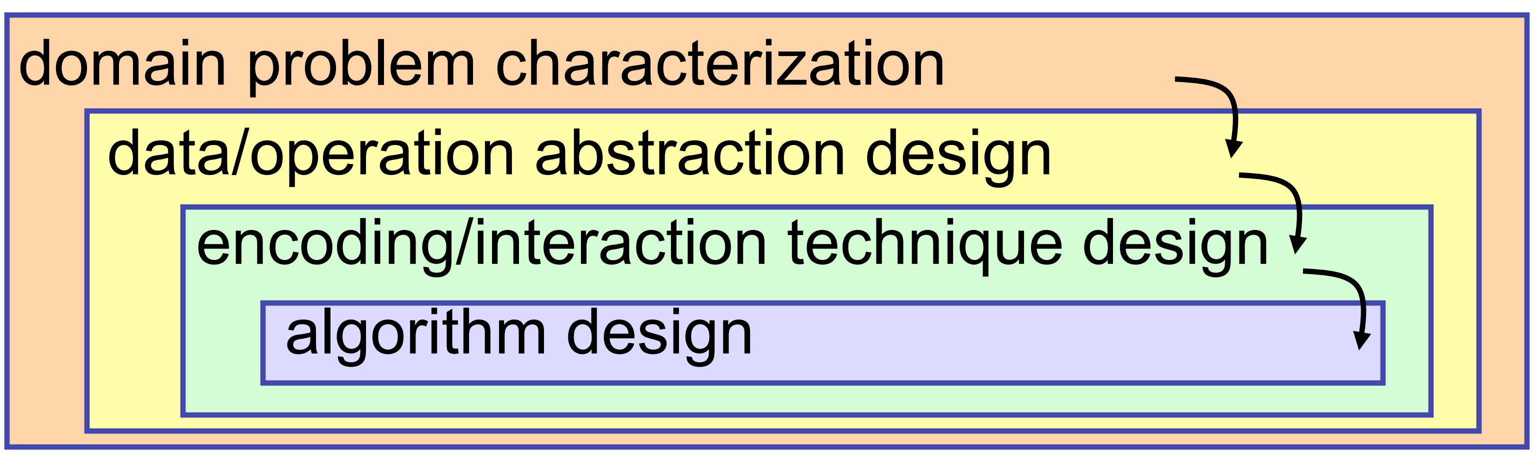
justify encoding/interaction design
qualitative result image analysis
test on target users, get utility anecdotes

Flow map layout. Phan et al. InfoVis 2005.

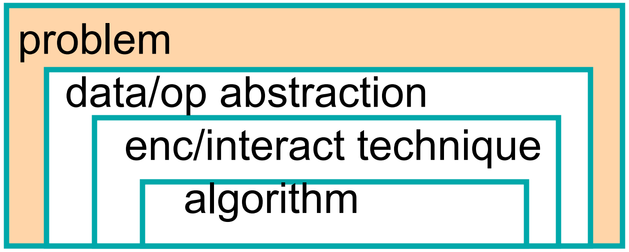
justify encoding/interaction design
computational complexity analysis
measure system time/memory
qualitative result image analysis

Nested levels in model

- output of **upstream** level
  input to **downstream** level
- challenge: upstream errors inevitably cascade
  - if poor abstraction choice made, even perfect technique and algorithm design will not solve intended problem

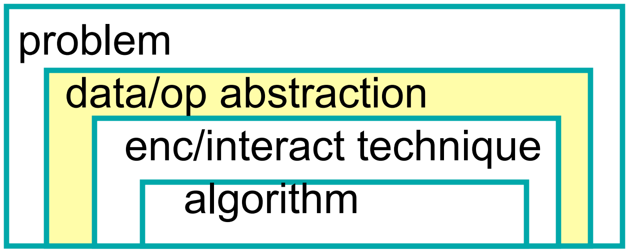


Characterizing domain problems



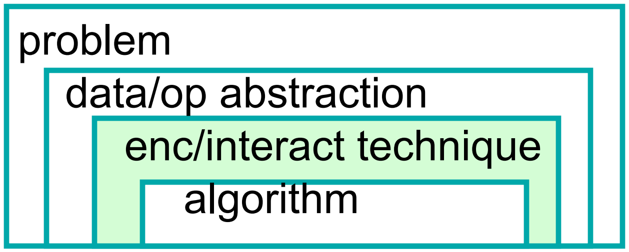
- tasks, data, workflow of target users
  - **problems**: tasks described in domain terms
  - requirements elicitation is notoriously hard

Designing data/operation abstraction



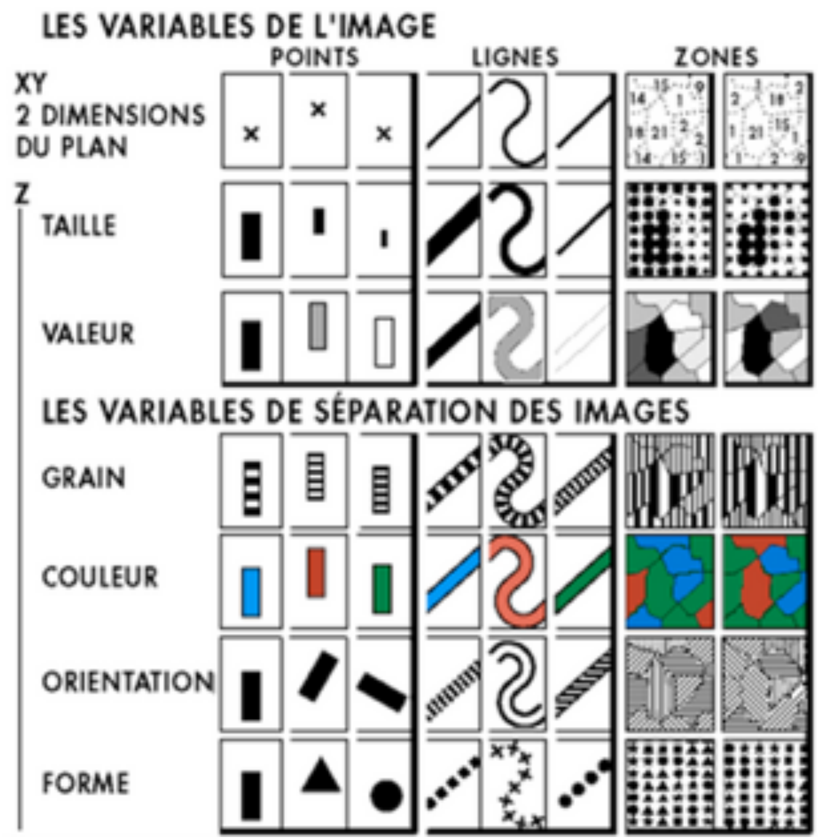
- mapping from domain vocabulary/concerns to abstraction
  - may require transformation!
- **data types**: data described in abstract terms
  - numeric tables, relational/network, spatial, ...
- **operations**: tasks described in abstract terms
  - generic
    - sorting, filtering, correlating, finding trends/outliers...
  - datatype-specific
    - path following through network...

Designing encoding,interaction techniques



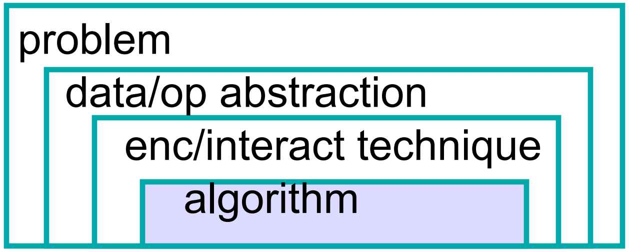
- visual encoding
  - marks, attributes, ...
  - extensive foundational work exists

- interaction
  - selecting, navigating, ordering, ...
  - significant guidance exists



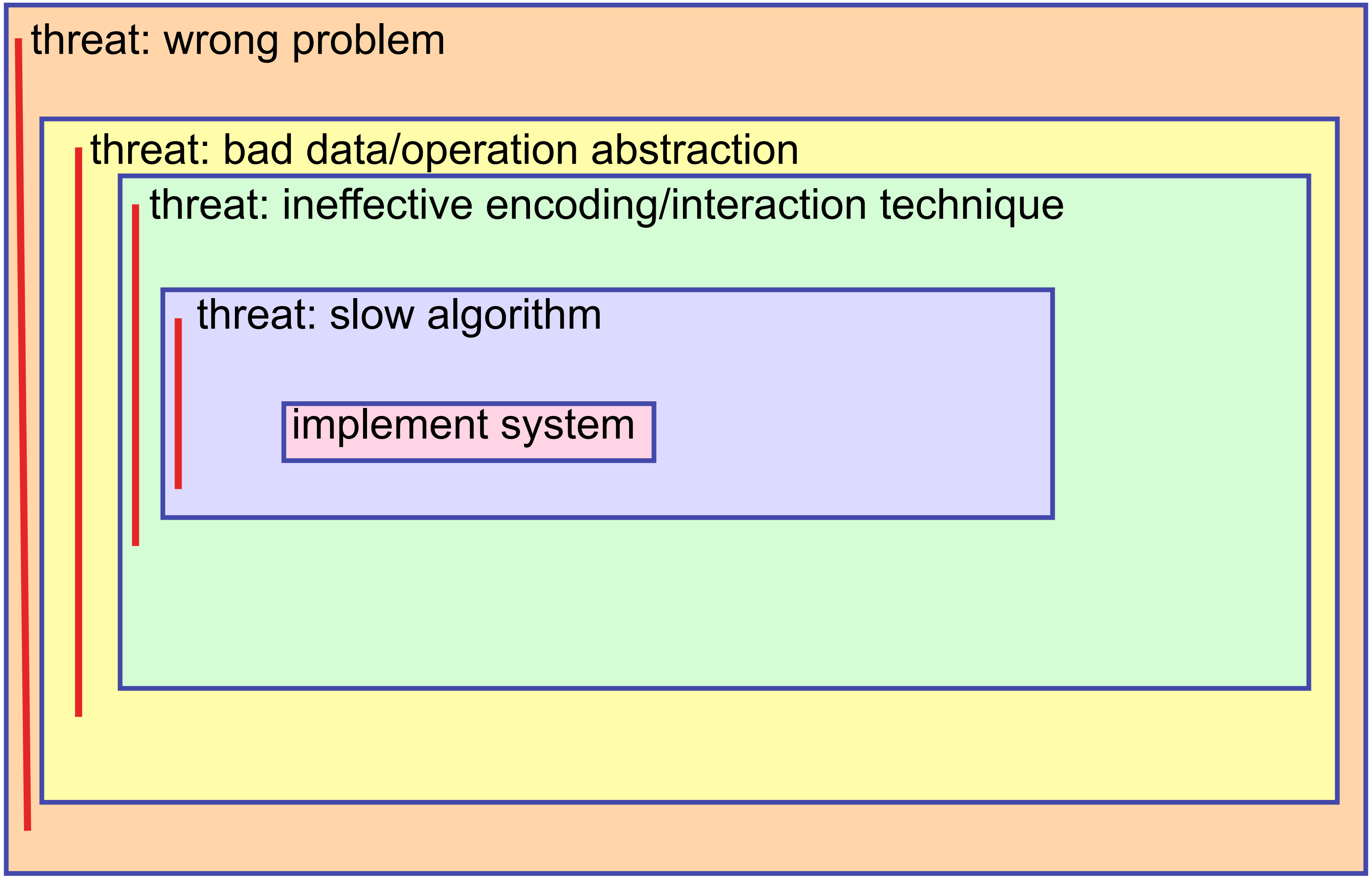
Semiology of Graphics. Jacques Bertin, Gauthier-Villars 1967, EHESS 1998

Designing algorithms



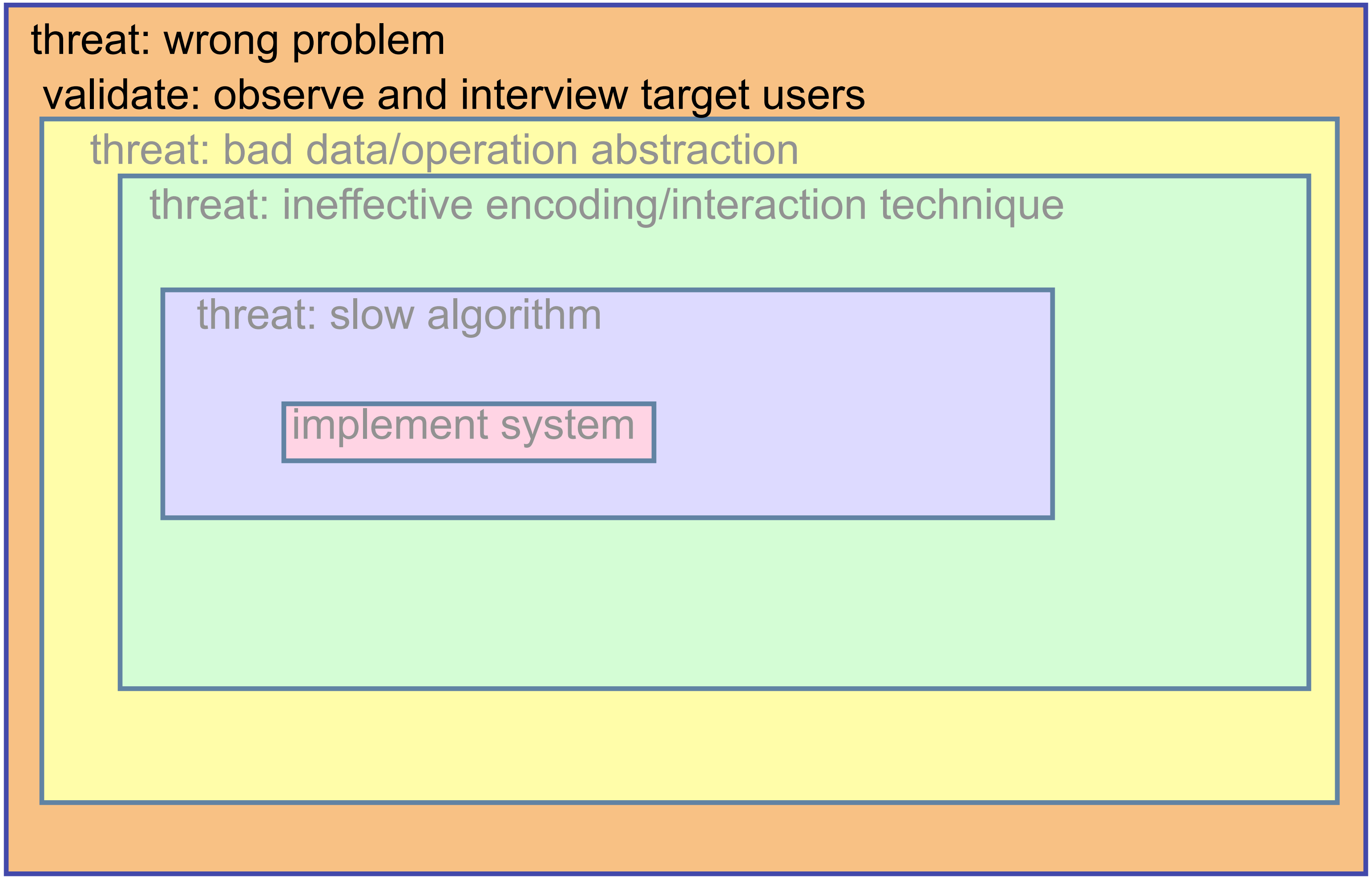
- well-studied computer science problem
  - create efficient algorithm given clear specification
  - no human-in-loop questions

Immediate vs. downstream validation



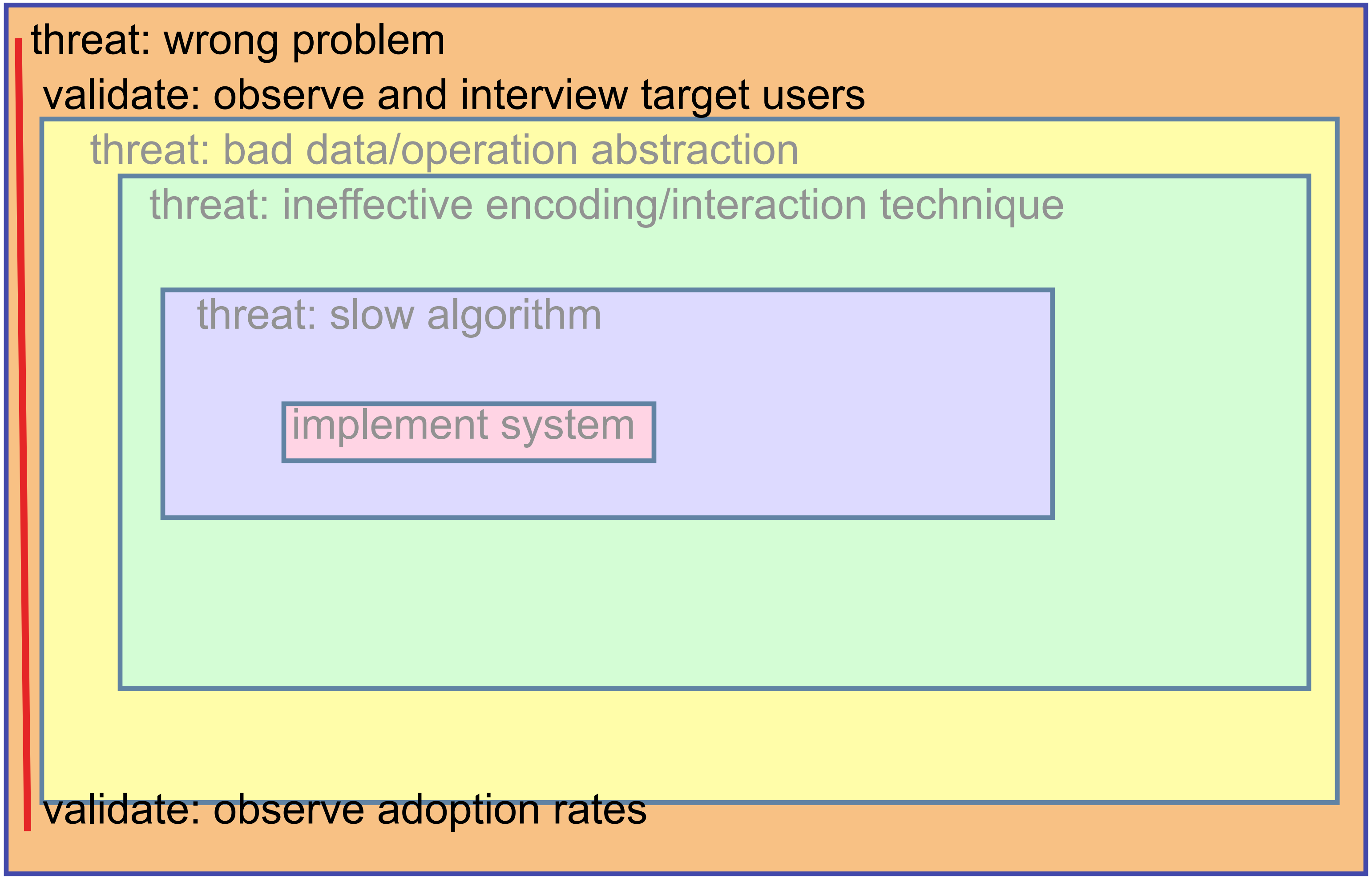
Domain problem validation

- immediate: ethnographic interviews/observations



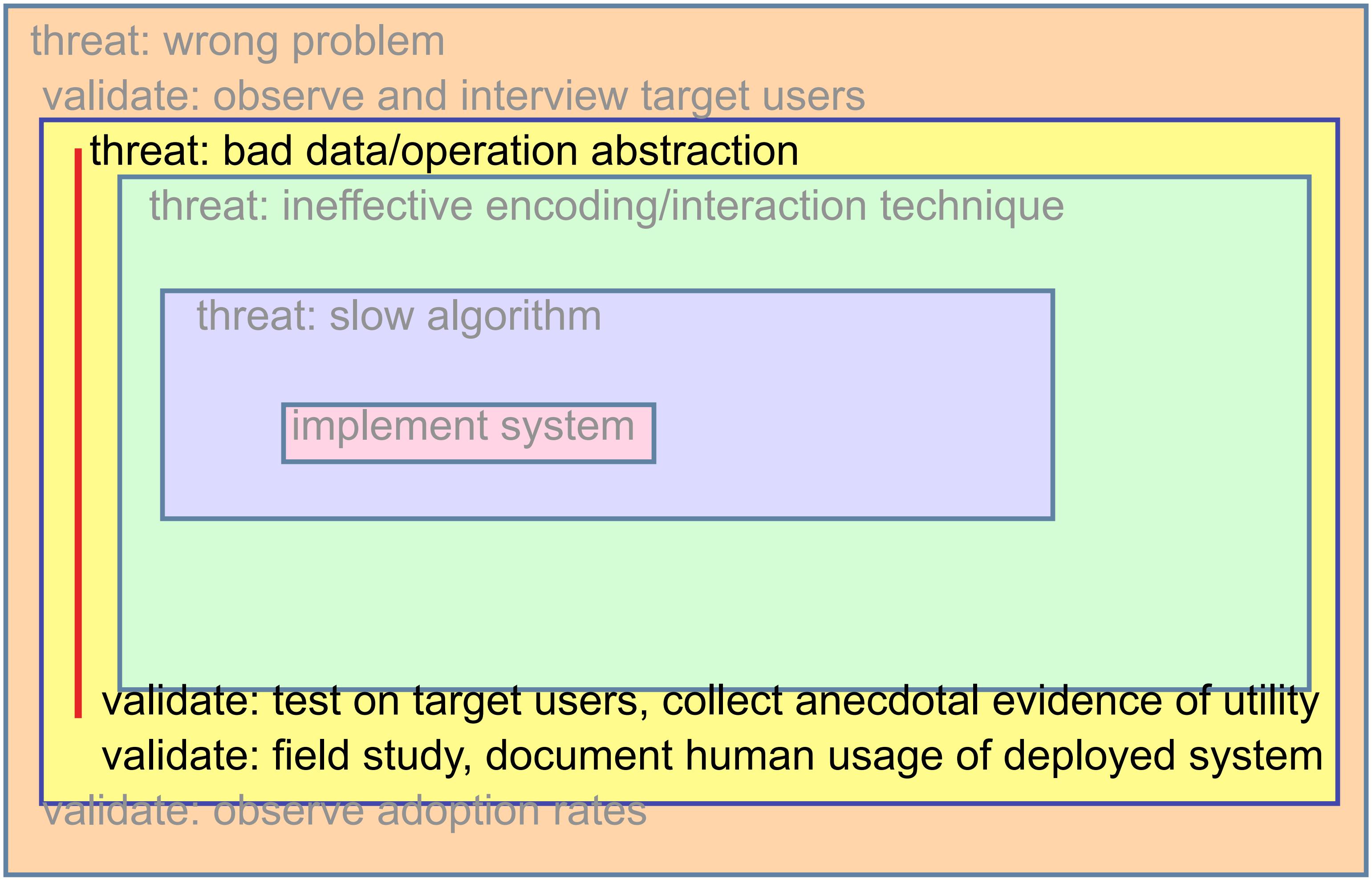
Domain problem validation

- downstream: adoption (weak but interesting signal)



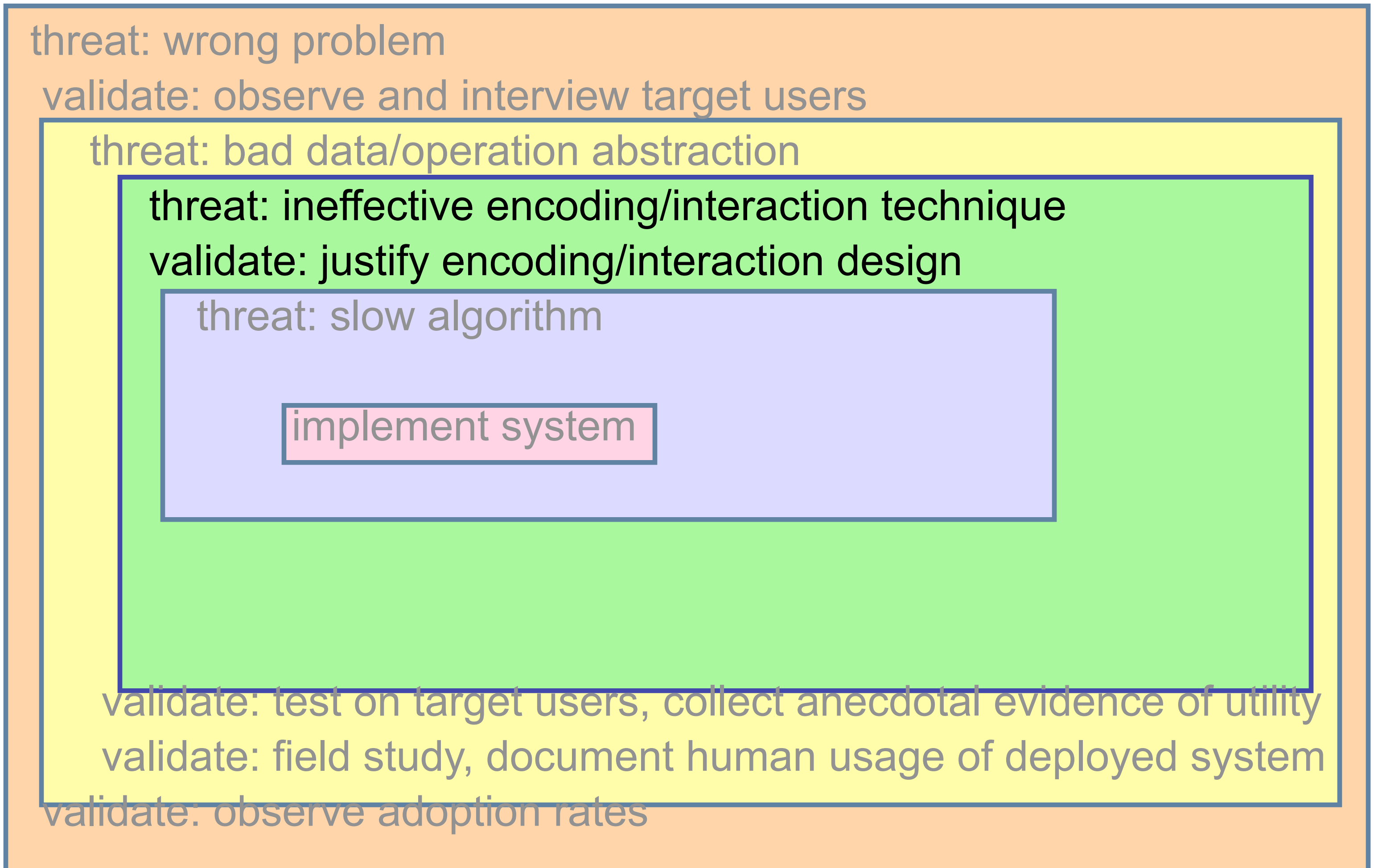
Abstraction validation

- downstream: can only test with target users doing real work



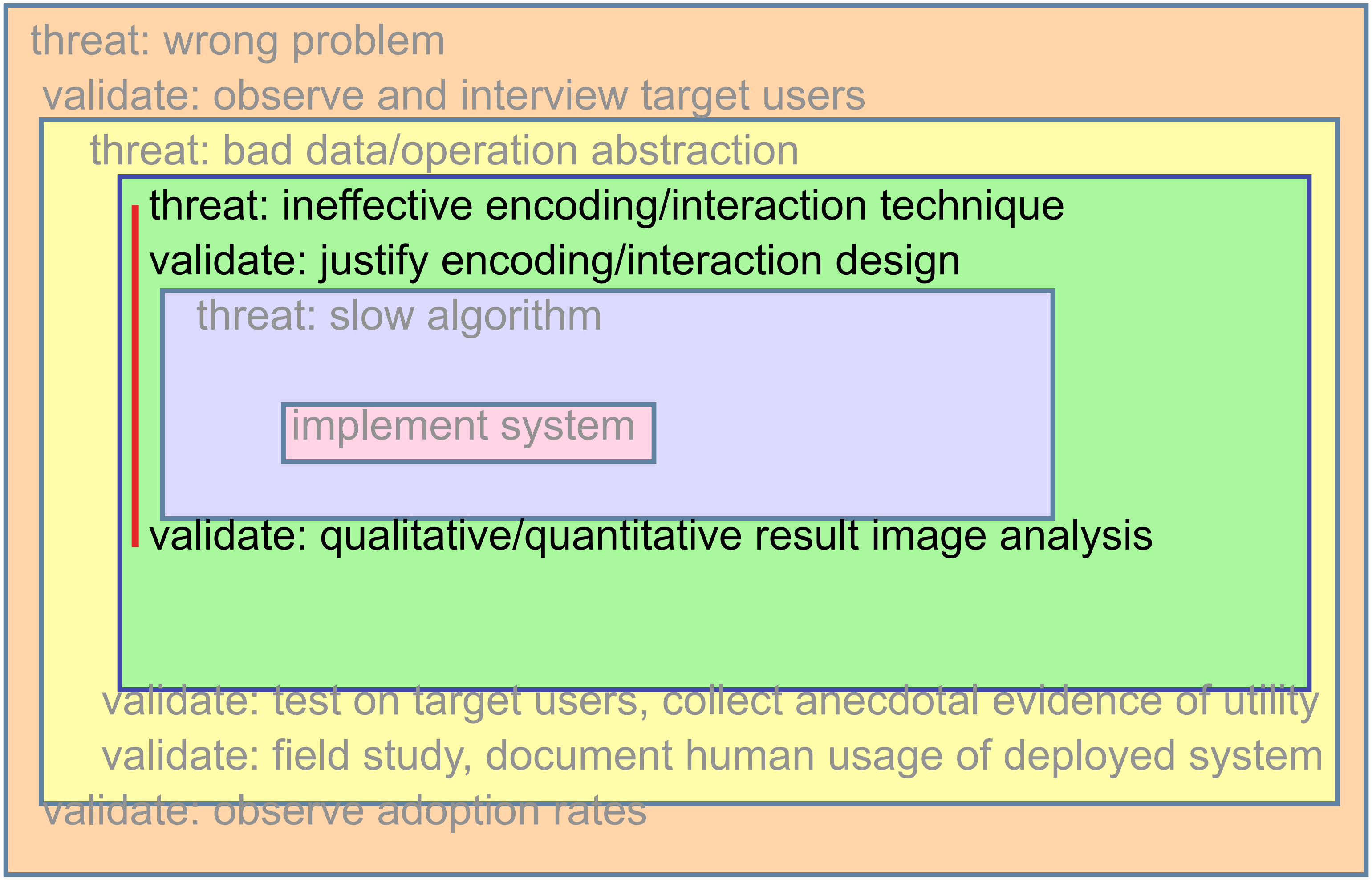
Encoding/interaction technique validation

- immediate: justification useful, but not sufficient - tradeoffs



Encoding/interaction technique validation

- downstream: discussion of result images very common



Encoding/interaction technique validation

- downstream: studies add another level of rigor (and time)

threat: wrong problem
validate: observe and interview target users
threat: bad data/operation abstraction
threat: ineffective encoding/interaction technique
validate: justify encoding/interaction design
threat: slow algorithm
implement system
validate: qualitative/quantitative result image analysis
validate: lab study, measure human time/errors for operation
validate: test on target users, collect anecdotal evidence of utility
validate: field study, document human usage of deployed system
validate: observe adoption rates

Encoding/interaction technique validation

- usability testing necessary for validity of downstream testing
  - not validation method itself!

threat: wrong problem
validate: observe and interview target users
threat: bad data/operation abstraction
threat: ineffective encoding/interaction technique
validate: justify encoding/interaction design
threat: slow algorithm
implement system
validate: qualitative/quantitative result image analysis
[test on any users, informal usability study]
validate: lab study, measure human time/errors for operation
validate: test on target users, collect anecdotal evidence of utility
validate: field study, document human usage of deployed system
validate: observe adoption rates

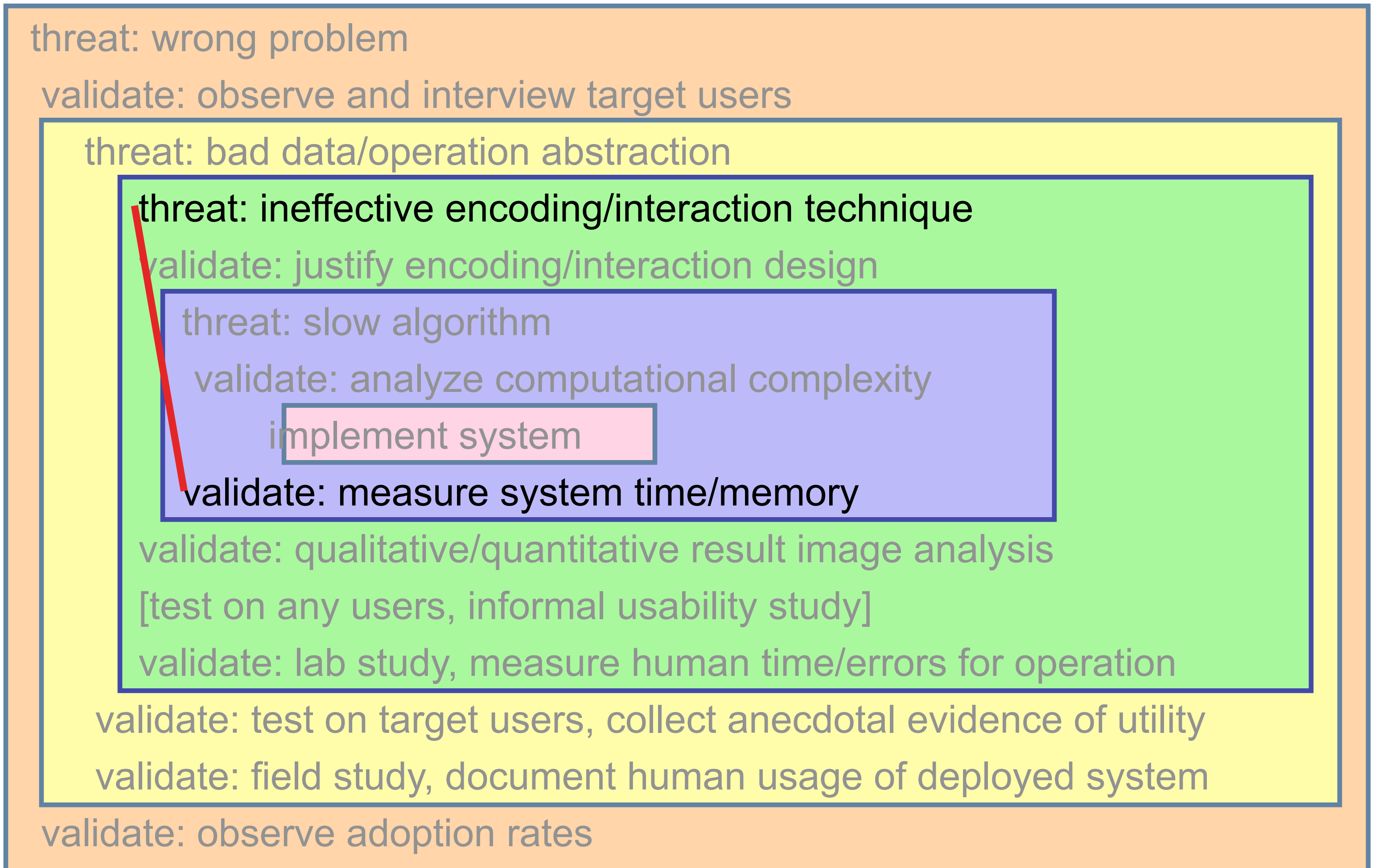
Algorithm validation

- immediate vs. downstream here clearly understood in CS

threat: wrong problem
validate: observe and interview target users
threat: bad data/operation abstraction
threat: ineffective encoding/interaction technique
validate: justify encoding/interaction design
threat: slow algorithm
validate: analyze computational complexity
implement system
validate: measure system time/memory
validate: qualitative/quantitative result image analysis
[test on any users, informal usability study]
validate: lab study, measure human time/errors for operation
validate: test on target users, collect anecdotal evidence of utility
validate: field study, document human usage of deployed system
validate: observe adoption rates

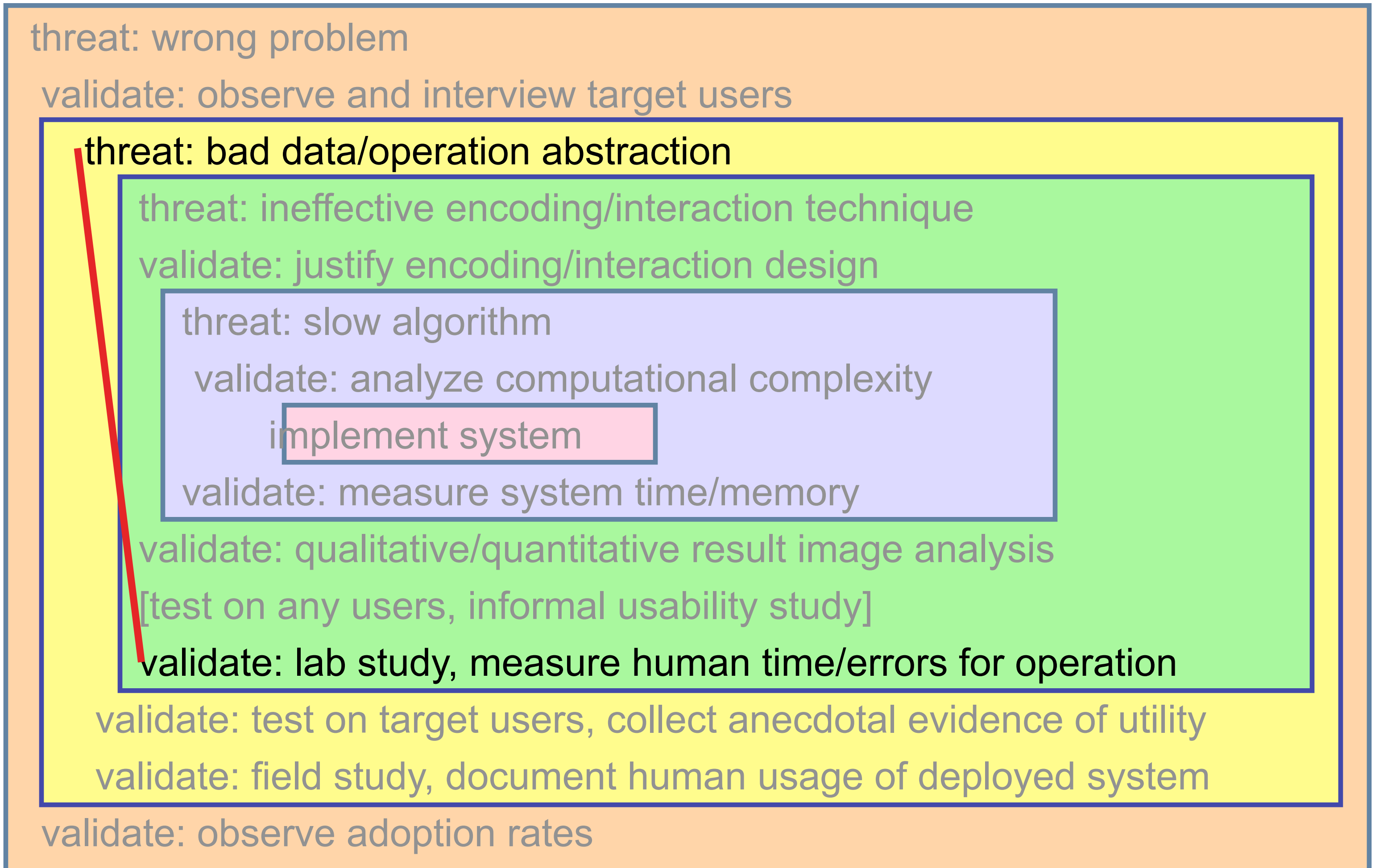
Avoid mismatches

- can't validate encoding with wallclock timings



Avoid mismatches

- can’t validate abstraction with lab study



Single paper would include only subset

- can't do all for same project
  - not enough space in paper or time to do work

threat: wrong problem

validate: observe and interview target users

threat: bad data/operation abstraction

threat: ineffective encoding/interaction technique

validate: justify encoding/interaction design

threat: slow algorithm

validate: analyze computational complexity

implement system

validate: measure system time/memory

validate: qualitative/quantitative result image analysis

[test on any users, informal usability study]

validate: lab study, measure human time/errors for operation

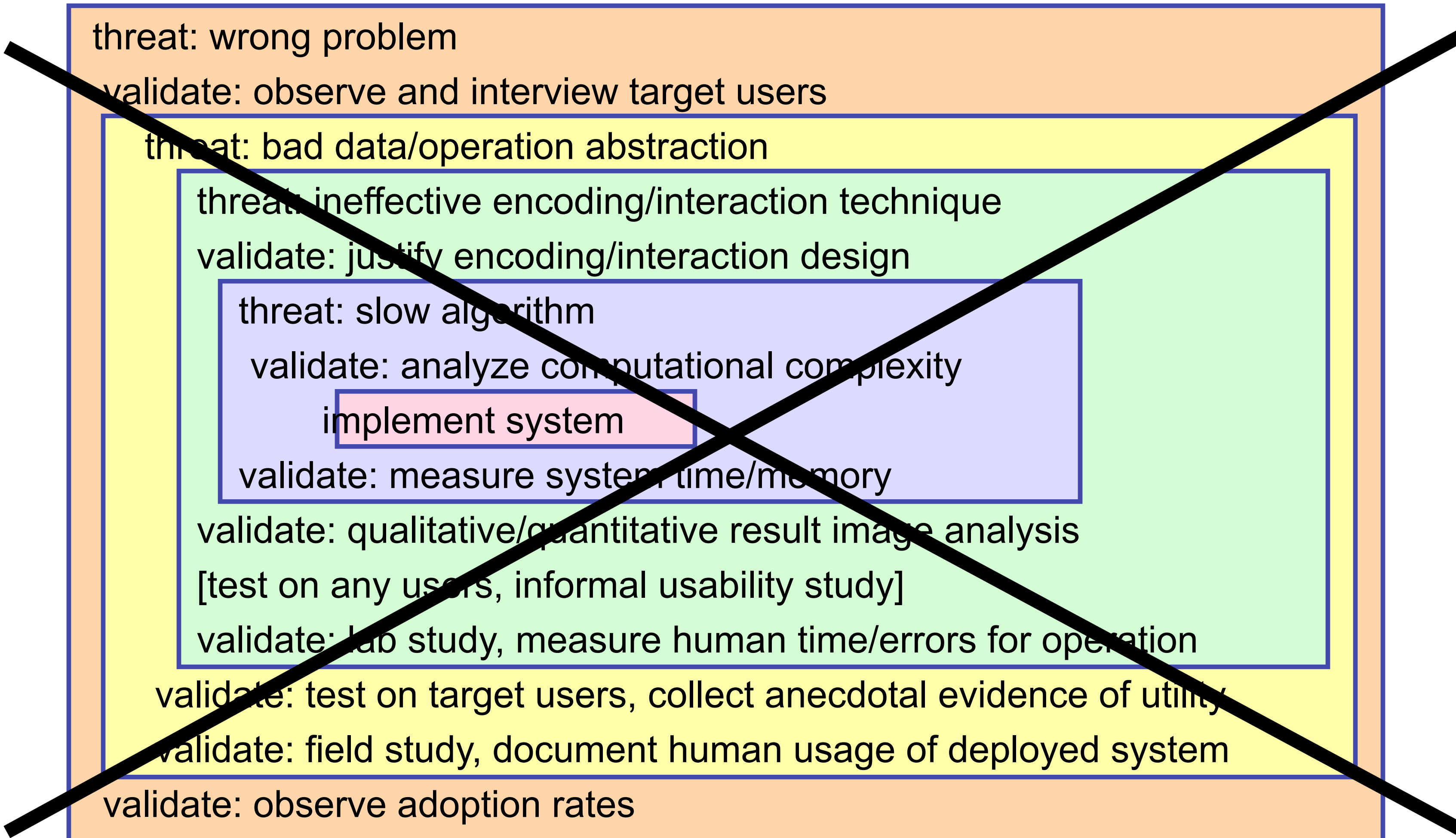
validate: test on target users, collect anecdotal evidence of utility

validate: field study, document human usage of deployed system

validate: observe adoption rates

Single paper would include only subset

- pick validation method according to contribution claims

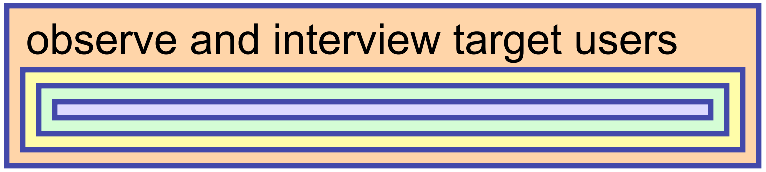


Recommendations: authors

- explicitly state level of contribution claim(s)
- explicitly state assumptions for levels upstream of paper focus
  - just one sentence + citation may suffice
- goal: literature with clearer interlock between papers
  - better unify problem-driven and technique-driven work

Recommendation: publication venues

- we need more problem characterization
  - ethnography, requirements analysis
- as part of paper, and as full paper
  - now full papers relegated to CHI/CSCW
    - does not allow focus on central vis concerns
  - **legitimize ethnographic “orange-box” papers!**



Limitations

- oversimplification
- not all forms of user studies addressed
- infovis-oriented worldview
- are these levels the right division?

Why EVALUATE?

Mike Gleicher
BELIV'12

GUIDE

How well do you inform
the audience to do some thing?

CONTEXT

PERSUADE

How well do you convince
the audience of some thing

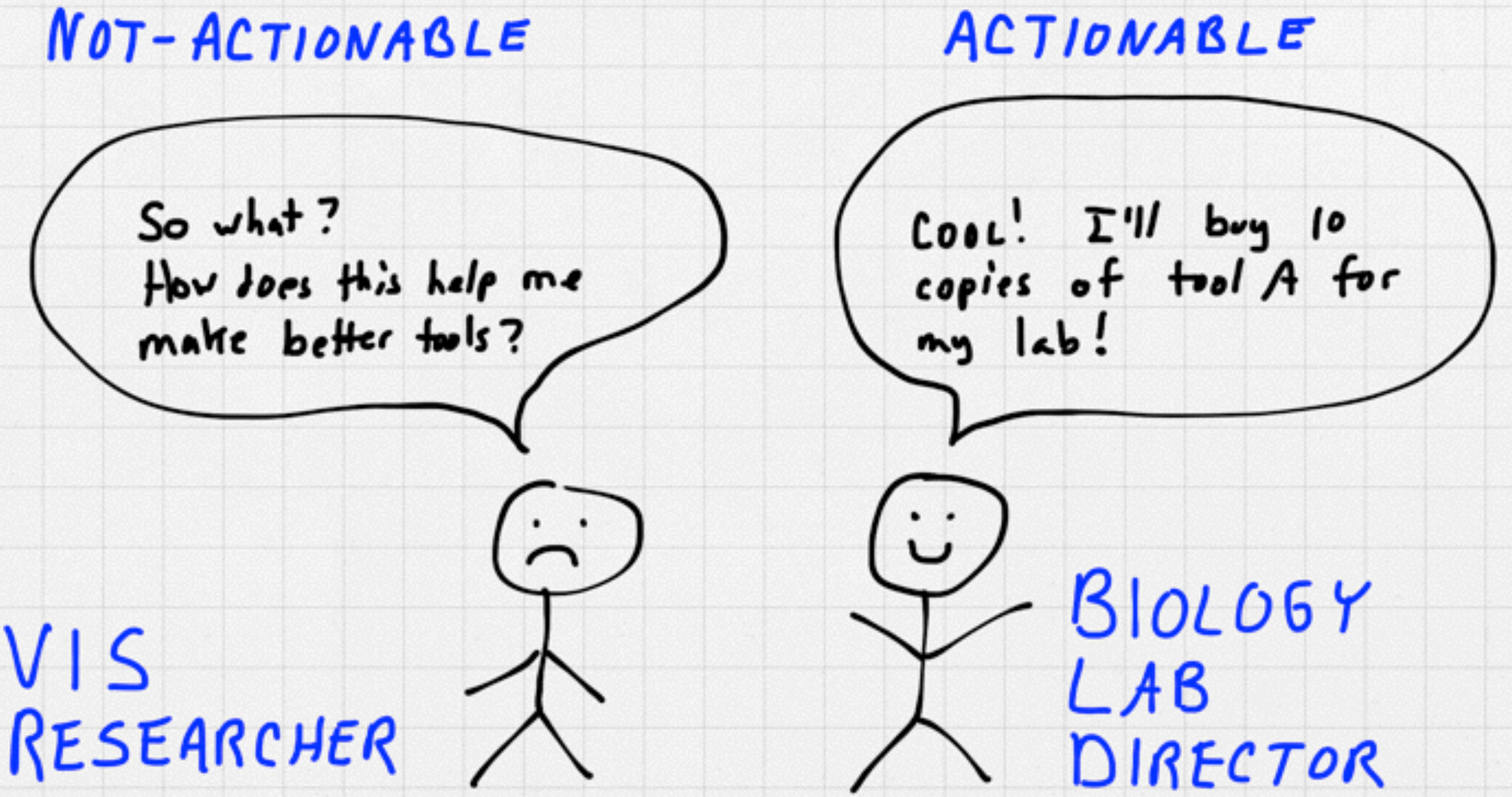
how ACTIONABLE and how PERSUASIVE depends on CONTEXT

SCIENTISTS USING TOOL A MAKE MORE DISCOVERIES THAN USING TOOL B

NOT-ACTIONABLE

ACTIONABLE

SCIENTISTS USING TOOL A MAKE MORE DISCOVERIES THAN USING TOOL B

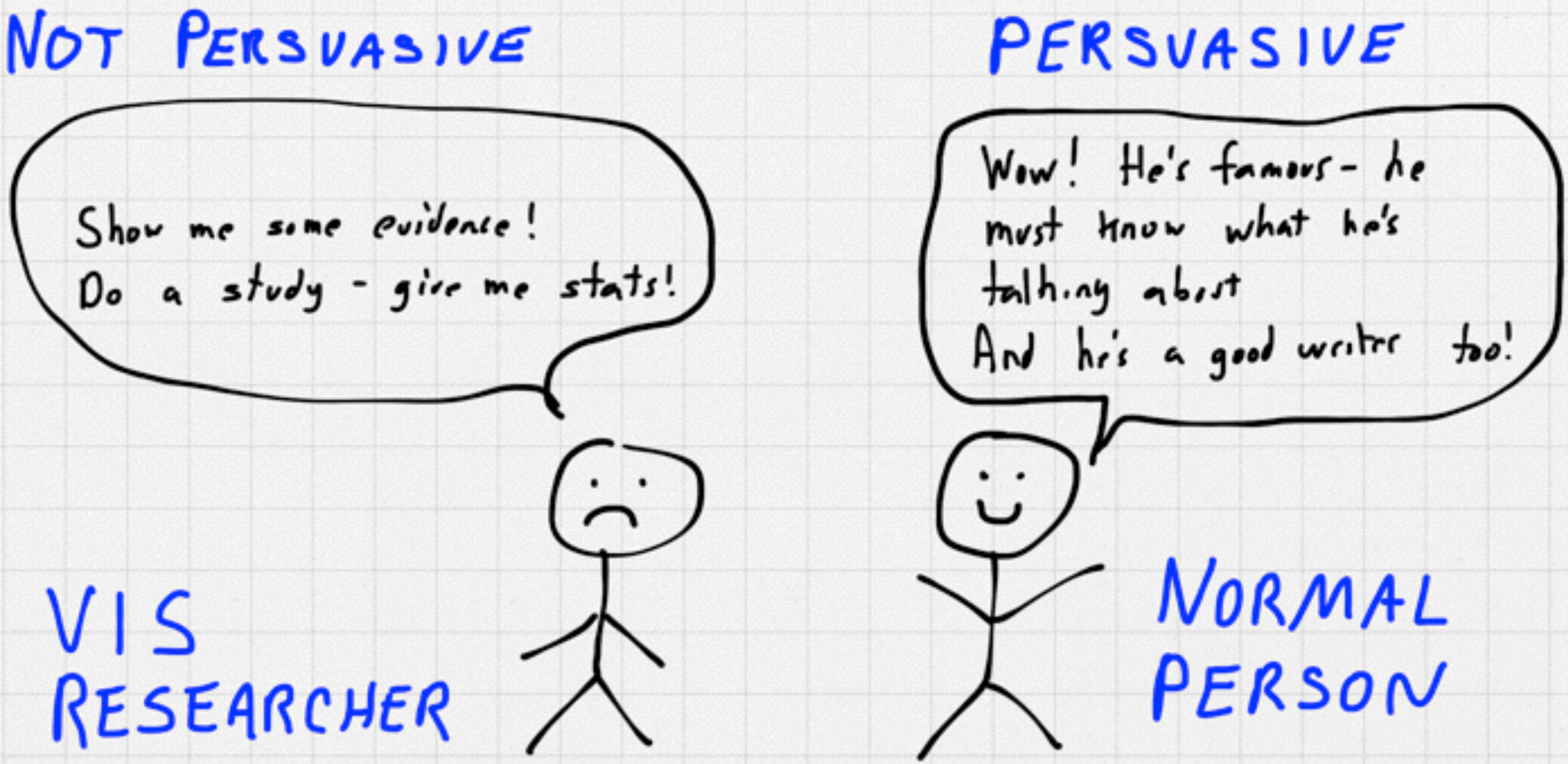


A PUNDIT ASSERTS MINIMALISM IS GOOD IN A SELF-PUBLISHED BOOK

NOT PERSUASIVE

PERSUASIVE

A PUNDIT ASSERTS MINIMALISM IS GOOD IN A SELF-PUBLISHED BOOK



EVALUATING EVALUATIONS

How well do they guide?

How well do they persuade?

Making GOOD Evaluations

Make it Actionable

make it Persuasive

To the target audience

HOW TO MAKE PERSUASIVE EVALUATIONS?

MEASURE THE RIGHT THINGS
DESIGN GOOD EXPERIMENTS

REPORT IT IN A WAY THAT CONVINCES THE AUDIENCE

Sophisticated methods may be harder to report

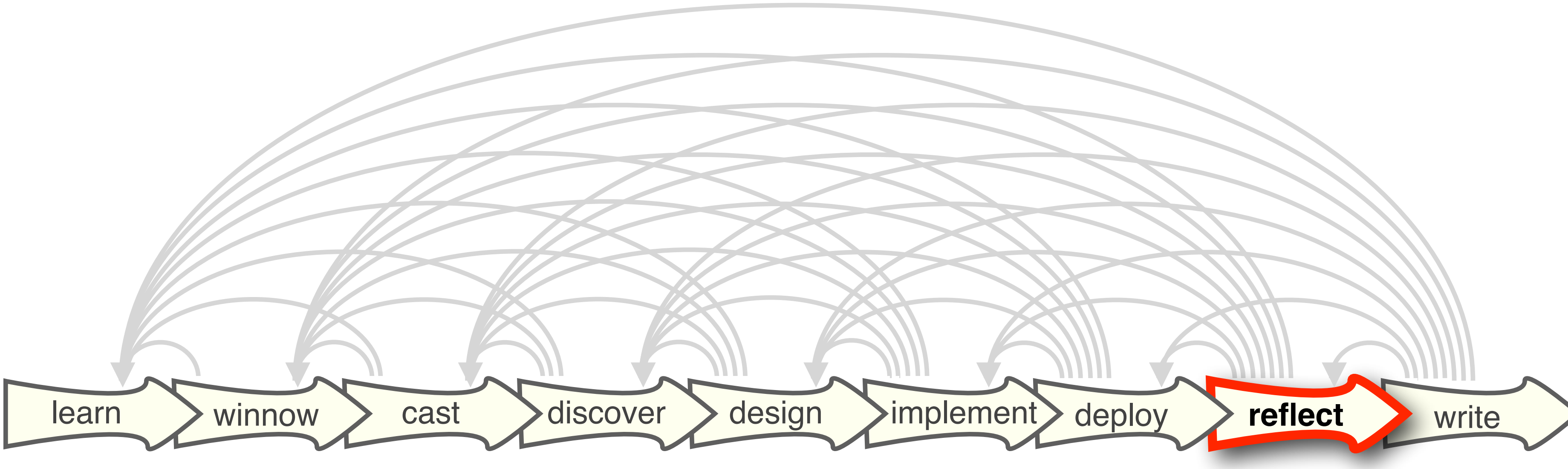
How to make ACTIONABLE evaluations ?

?

The Four-Level Nested Model Revisited: Blocks and Guidelines

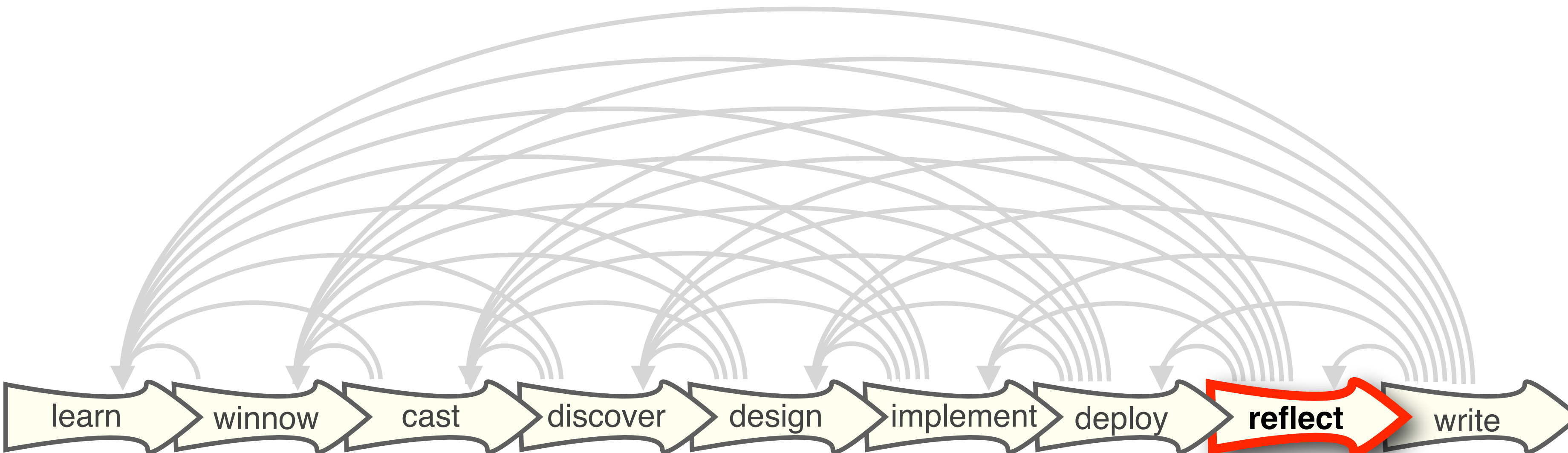
Miriah Meyer, Michael Selmair, Tamara Munzner

BELIV'12



Design Study Methodology: Reflections for the Trenches and the Stacks.

M. Sedlmair, M. Meyer, T. Munzner, IEEE TVCG (Proceedings of InfoVis 2012).



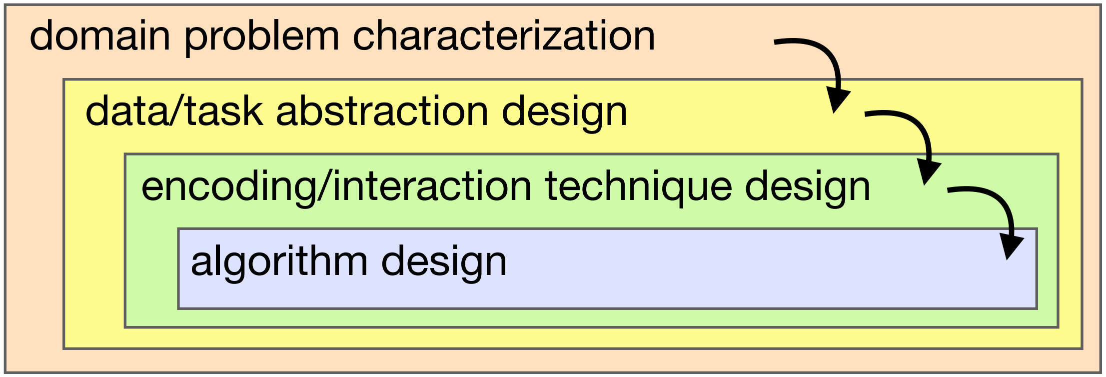
confirm | refine | reject | propose

guidelines

Design Study Methodology: Reflections for the Trenches and the Stacks.

M. Sedlmair, M. Meyer, T. Munzner, IEEE TVCG (Proceedings of InfoVis 2012).

NESTED MODEL

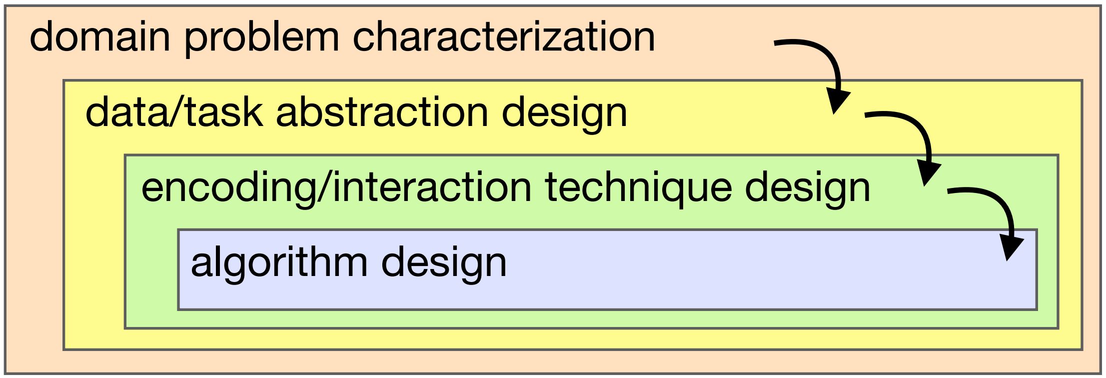


Munzner 2009

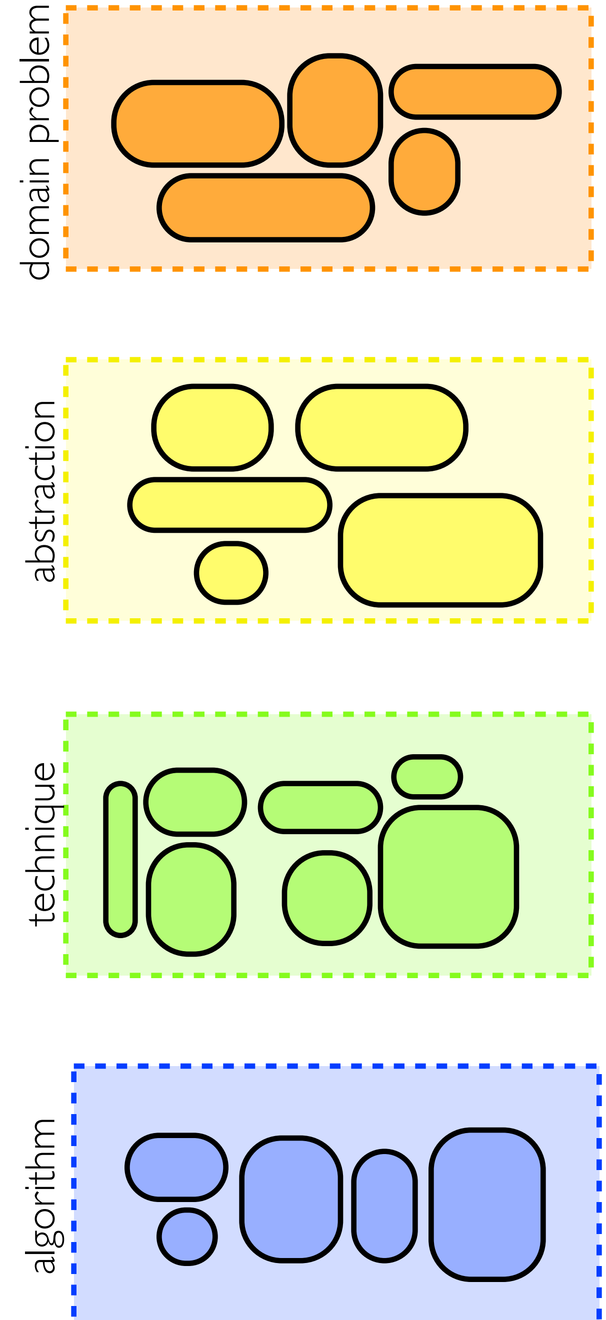
NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

NESTED MODEL



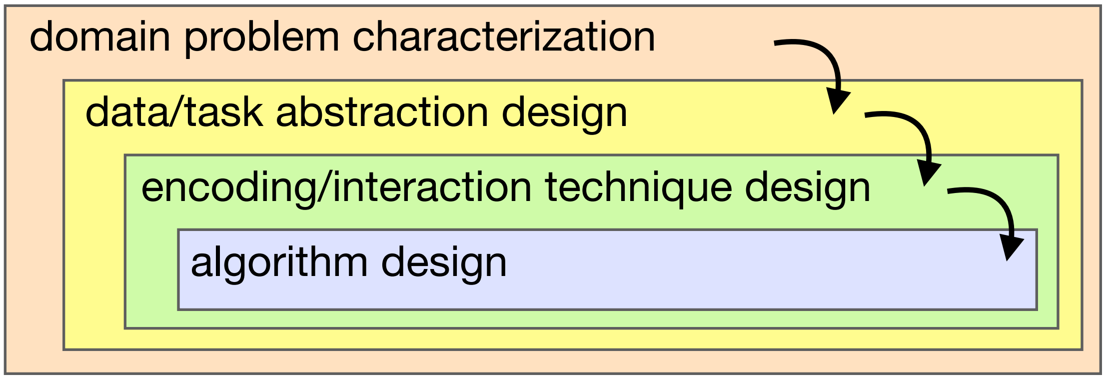
Munzner 2009



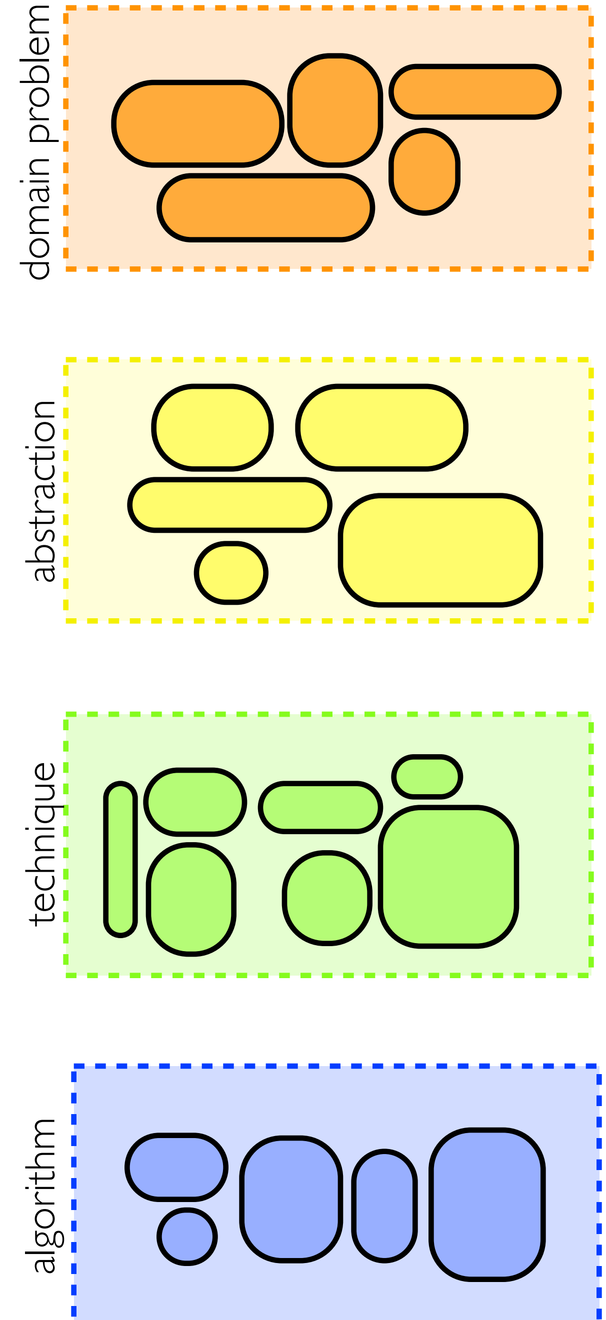
NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

NESTED MODEL



Munzner 2009



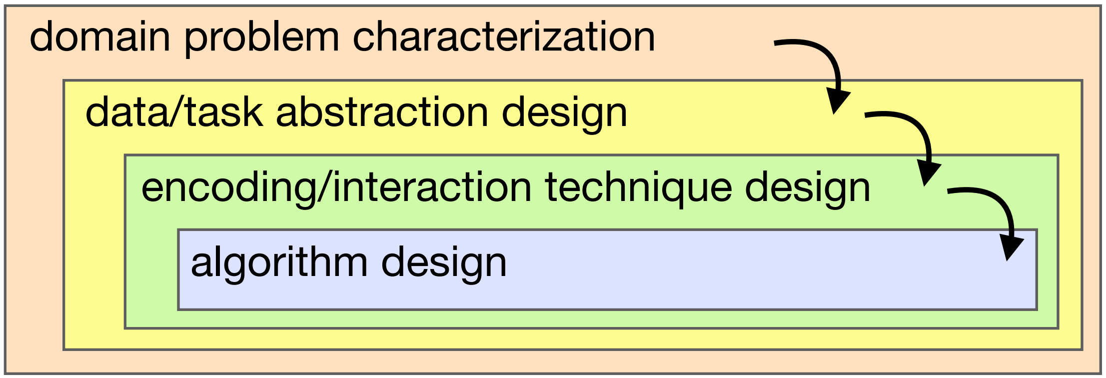
blocks

outcome of a design decision

NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

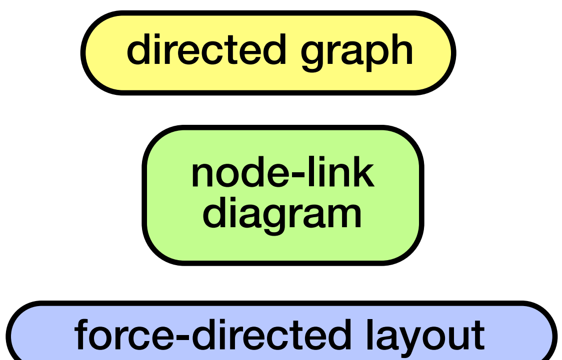
NESTED MODEL

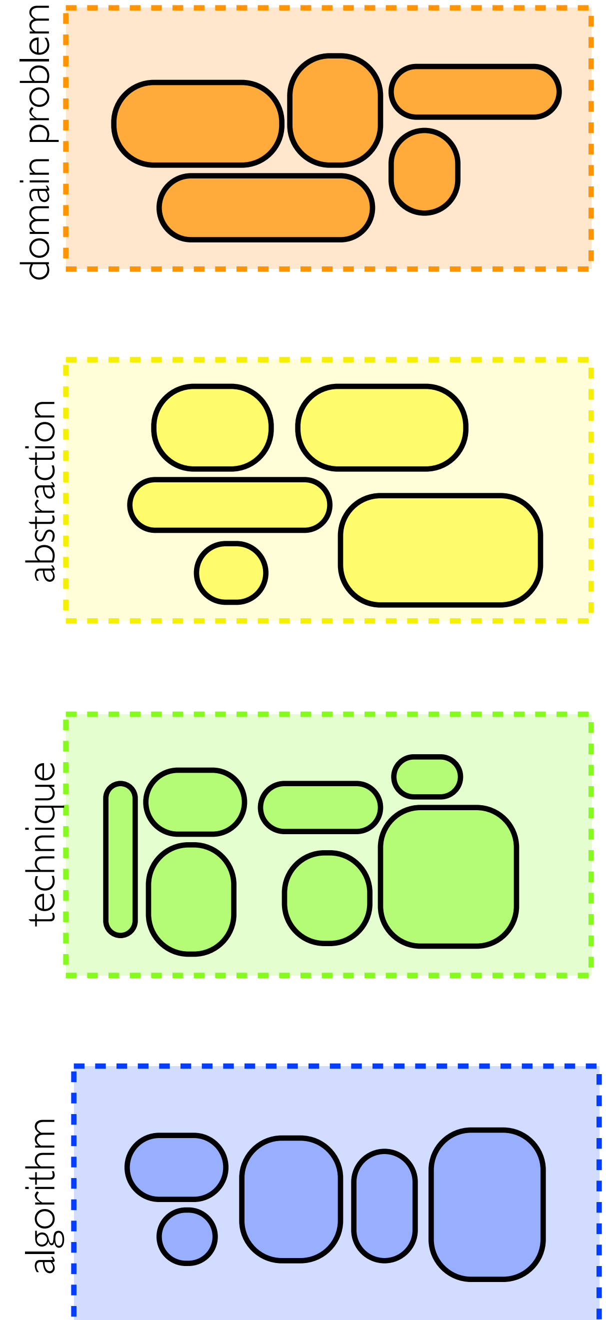


Munzner 2009

blocks

outcome of a design decision

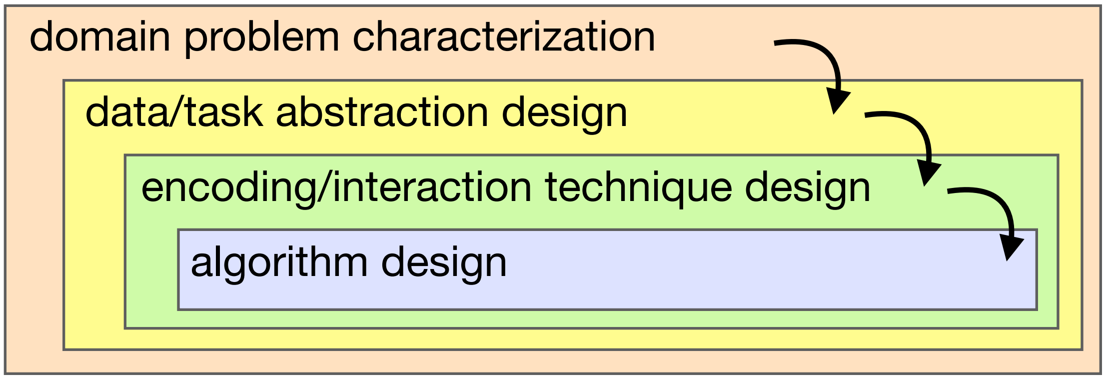




NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

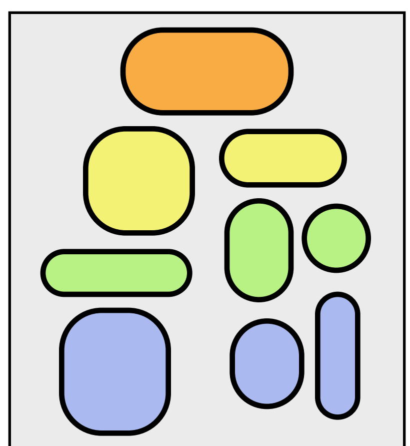
NESTED MODEL

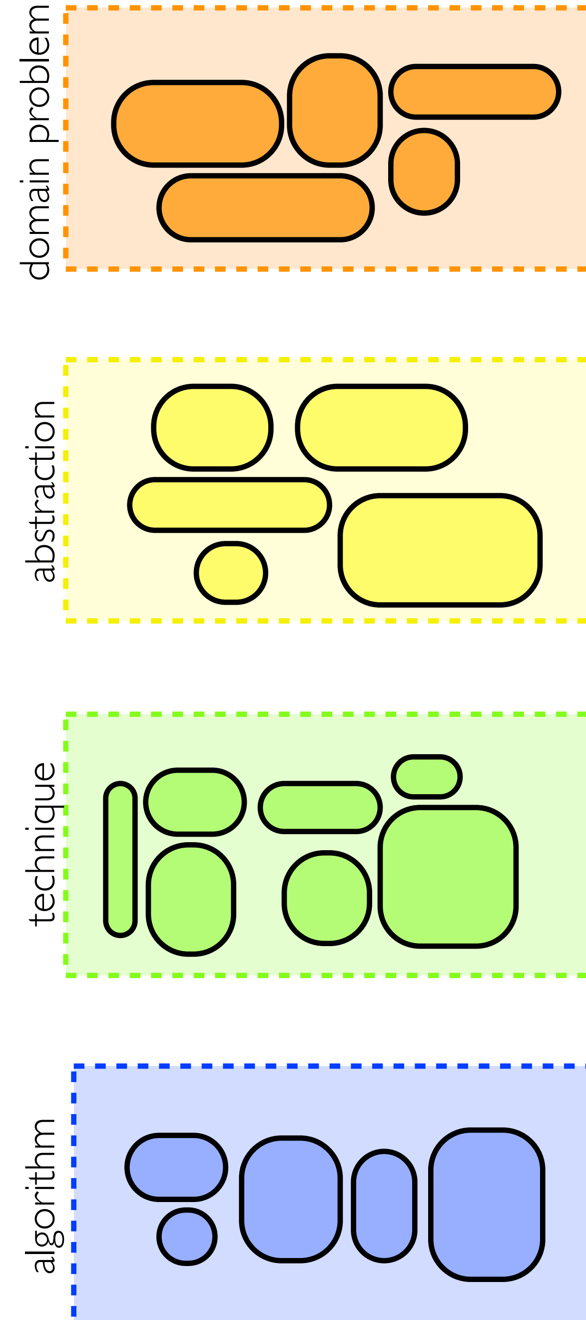


Munzner 2009

blocks

outcome of a design decision

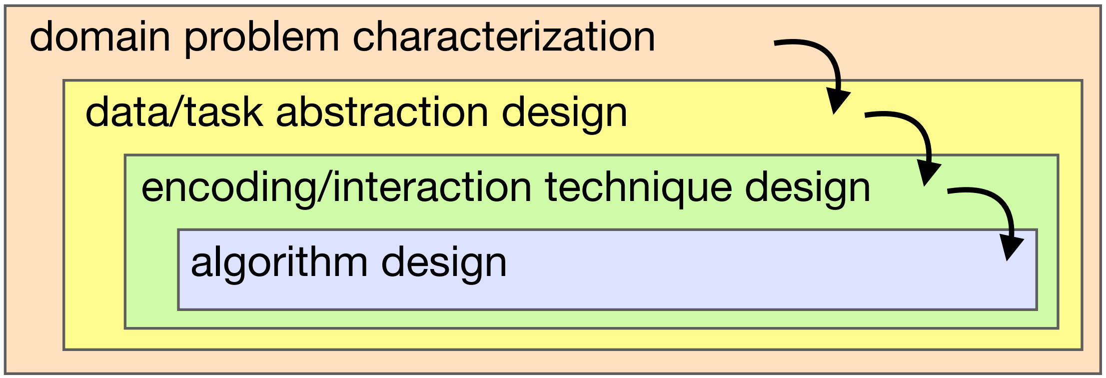




NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

NESTED MODEL

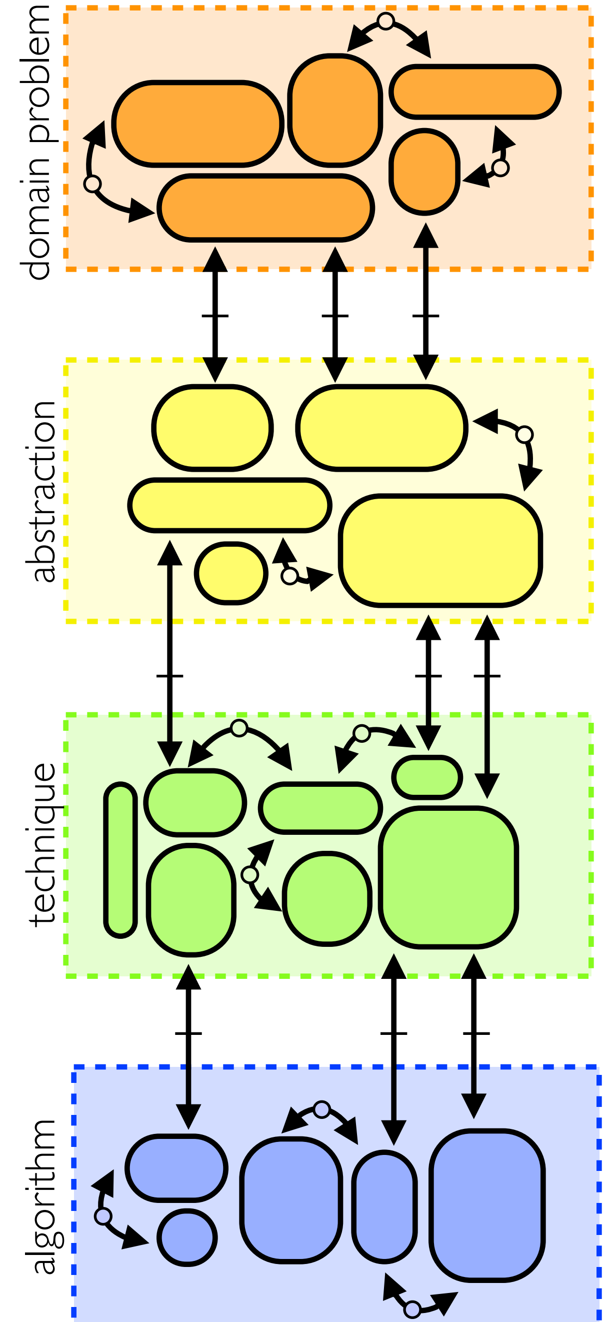


Munzner 2009

blocks

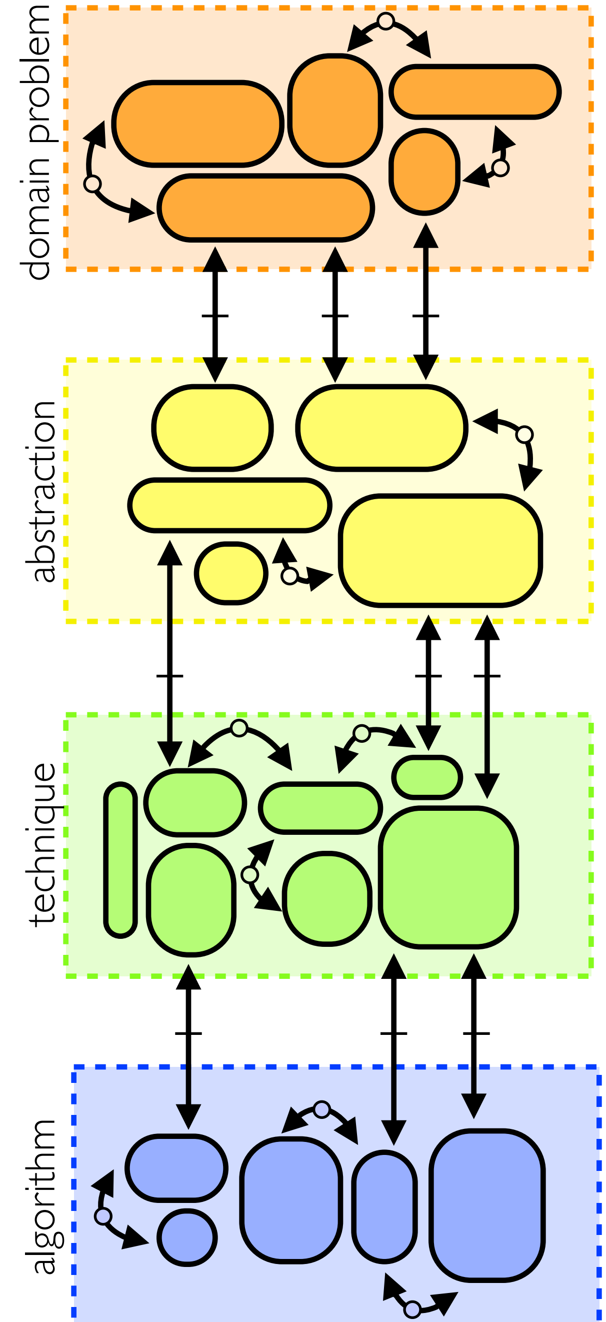
guidelines

statement about relationship between blocks

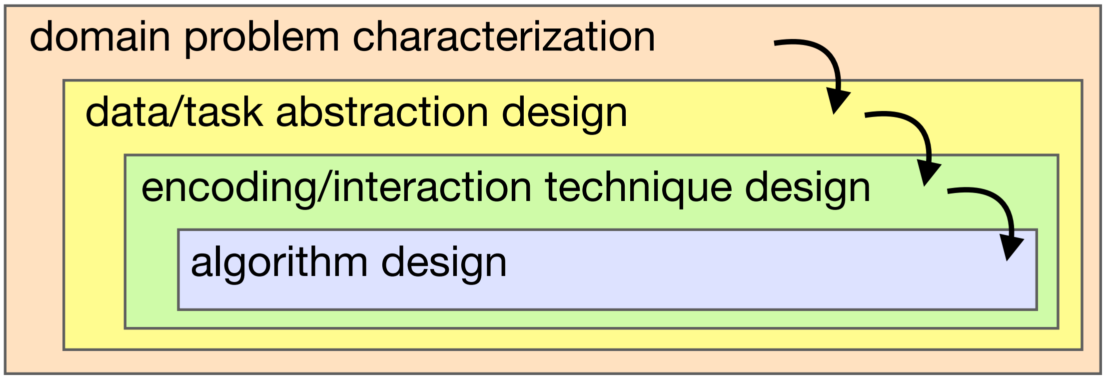


NESTED BLOCKS AND GUIDELINES

[Meyer 2013]



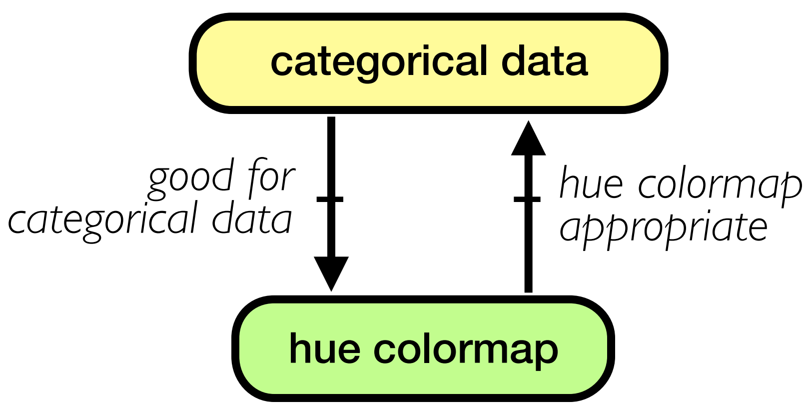
NESTED MODEL



Munzner 2009

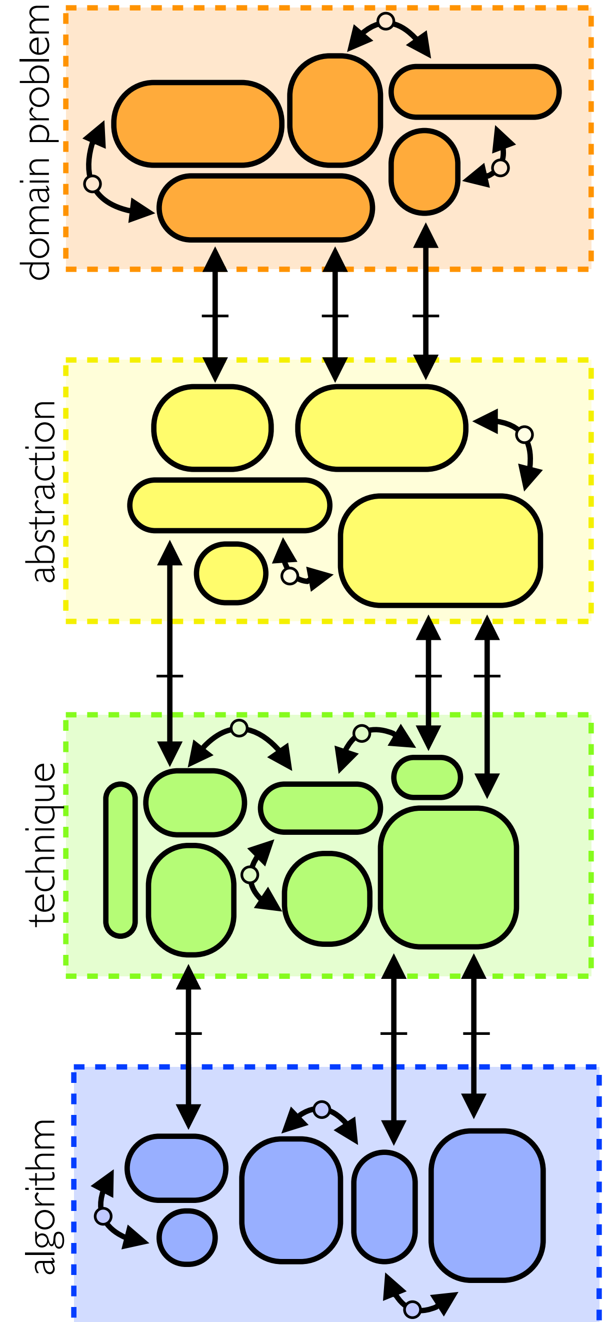
blocks

guidelines

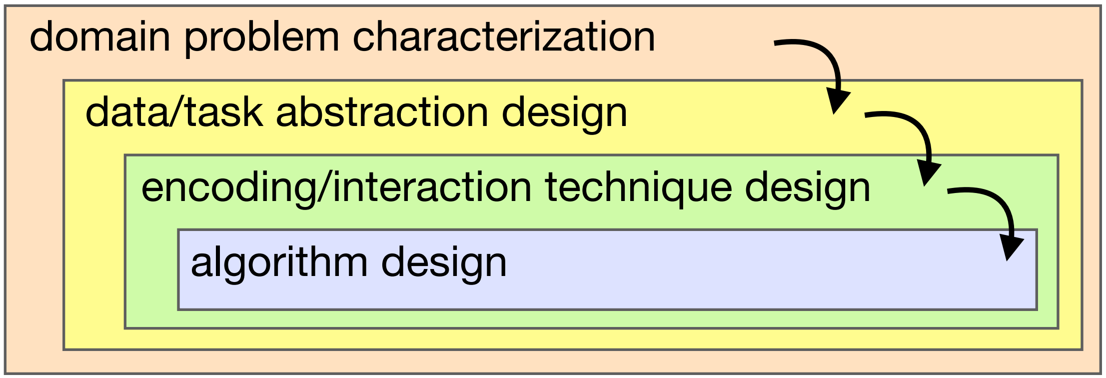


NESTED BLOCKS AND GUIDELINES

[Meyer 2013]



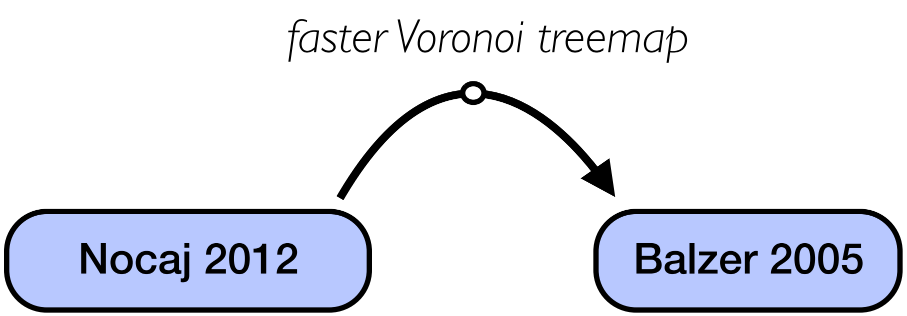
NESTED MODEL



Munzner 2009

blocks

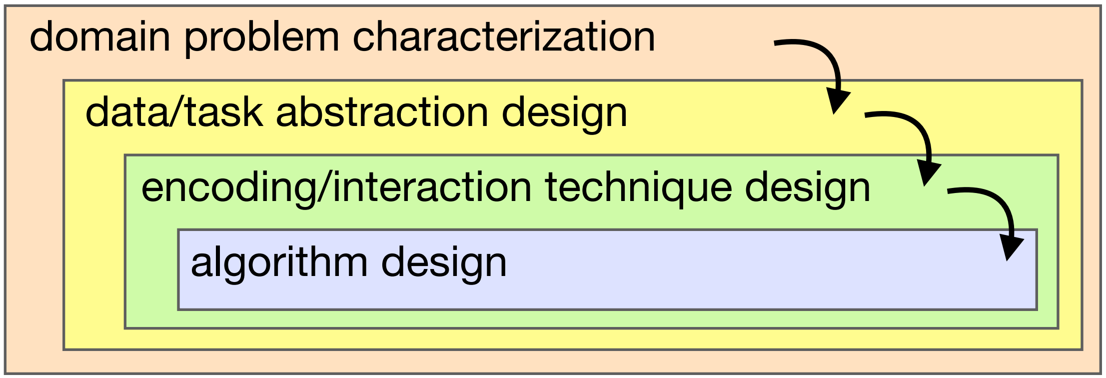
guidelines



NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

NESTED MODEL



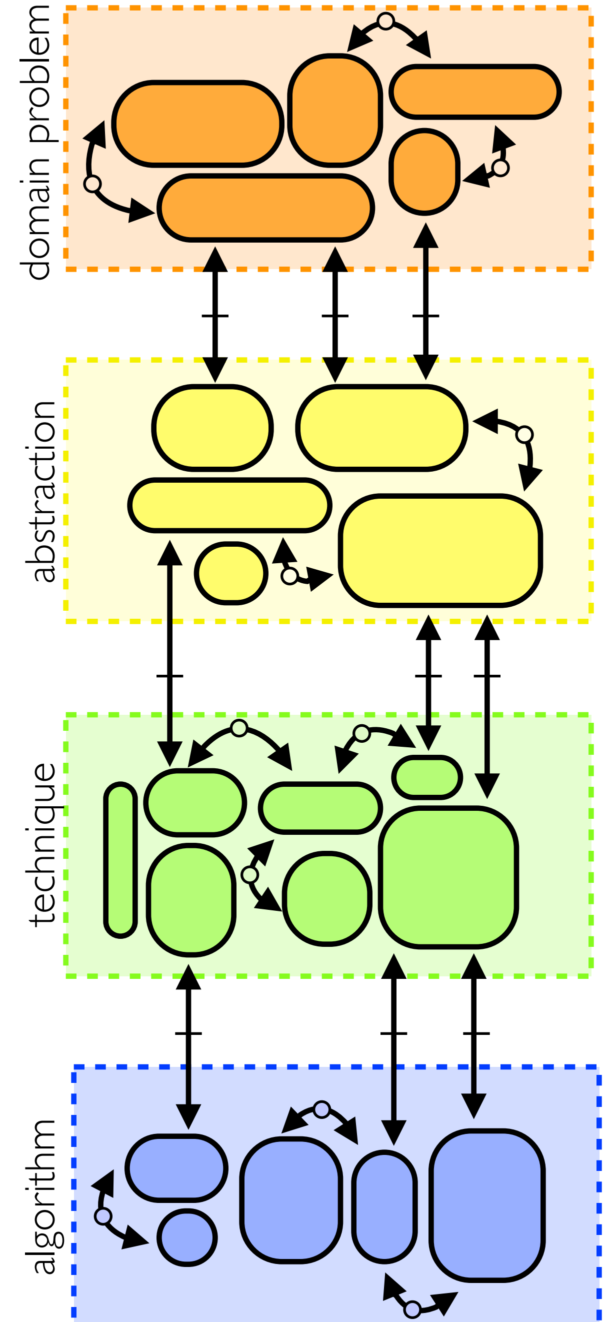
Munzner 2009

blocks

guidelines

between-level guideline

within-level guideline



Interactive Level-of-Detail Rendering of Large Graphs

Michael Zinsmaier, Ulrik Brandes, Oliver Deussen, and Hendrik Strobelt

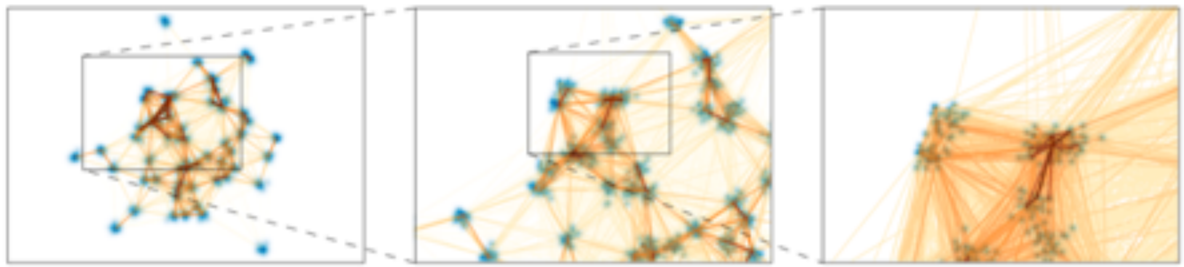


Fig. 1. Application of our visualization technique on a hierarchical data set, zooming from overview (left) to a region of interest (right). The density-based node aggregation field (blue color) guides edge aggregation (orange/red color) to reveal visual patterns at different levels of detail.

Abstract— We propose a technique that allows straight-line graph drawings to be rendered interactively with adjustable level of detail. The approach consists of a novel combination of edge cumulation with density-based node aggregation and is designed to exploit common graphics hardware for speed. It operates directly on graph data and does not require precomputed hierarchies or meshes. As proof of concept, we present an implementation that scales to graphs with millions of nodes and edges, and discuss several example applications.

Index Terms—Graph visualization, OpenGL, edge aggregation.

1 Introduction

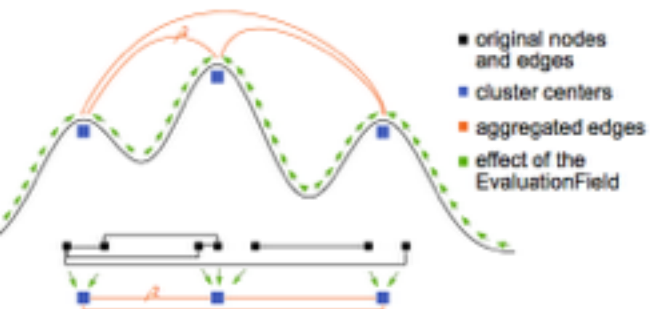
We present methods for the interactive visualization of large graphs. We say a graph is large if it fits into video memory but cannot be rendered as node link diagram without significant over-plotting, thus we define size relative to the computing environment. For the interactive exploration of such graphs fast node and edge aggregation is needed in combination with efficient rendering in different levels of detail (LOD). Both is presented in the following. Our techniques enable us to show graphs with up to $\sim 10^7$ nodes and up to $\sim 10^6$ edges at interactive rates.

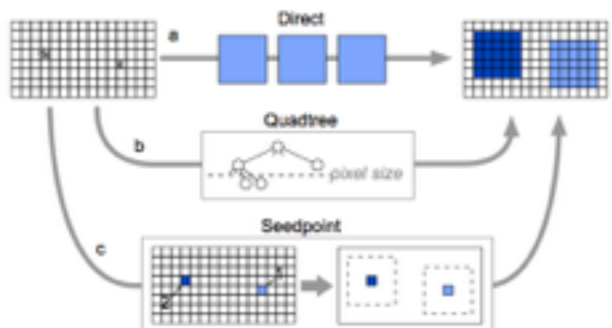
Lampe and Hauser [20] describe a method for rendering large graphs as density fields based on a GPU implementation of *Kernel Density Estimation (KDE)*. Our method extends their technique for node aggregation by a two-pass seed point rendering that significantly reduces geometry and scales to large graphs. Furthermore we present a fast edge aggregation method that derives start- and endpoints of ods is given in Section 3, performance considerations are discussed in Section 4. An interactive system based on the proposed techniques is described in Section 5. We present its interaction paradigms and some example applications. Finally, we summarize and propose future work in Section 6.

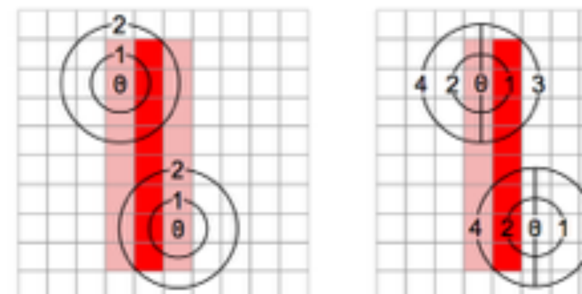
2 Related Work

We divide the problem of rendering large graphs on (comparatively) small displays into two main problems: dense regions of nodes and cluttering of edges. While the first is the general problem of dense point sets commonly faced in visualization and computer graphics, the second problem is more closely related to structure-aware methods from information visualization and graph drawing.

2.1 Node Visualization Methods

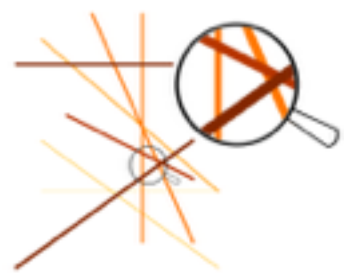






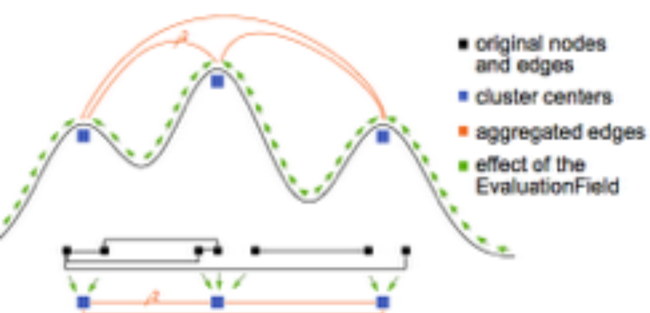
a) with hotspots

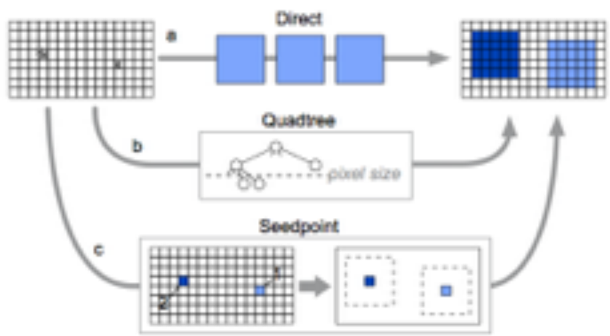
b) hotspots removed with angle separated rendering

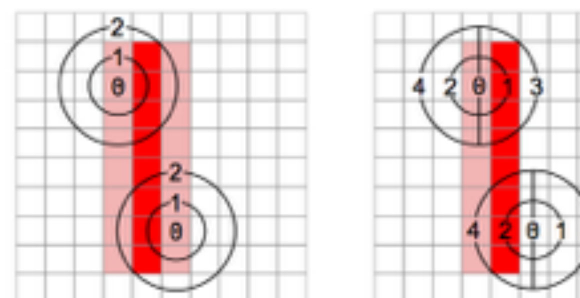


c) hotspots removed with angle separated rendering and color scaling corrected

- **LOD approach for rendering large graphs**
  - cluster nodes using GPU-based approach
  - aggregate edges
  - rendering issues
  - semantic zooming







a) with hotspots

b) hotspots removed with angle separated rendering

c) hotspots removed with angle separated rendering and color scaling corrected

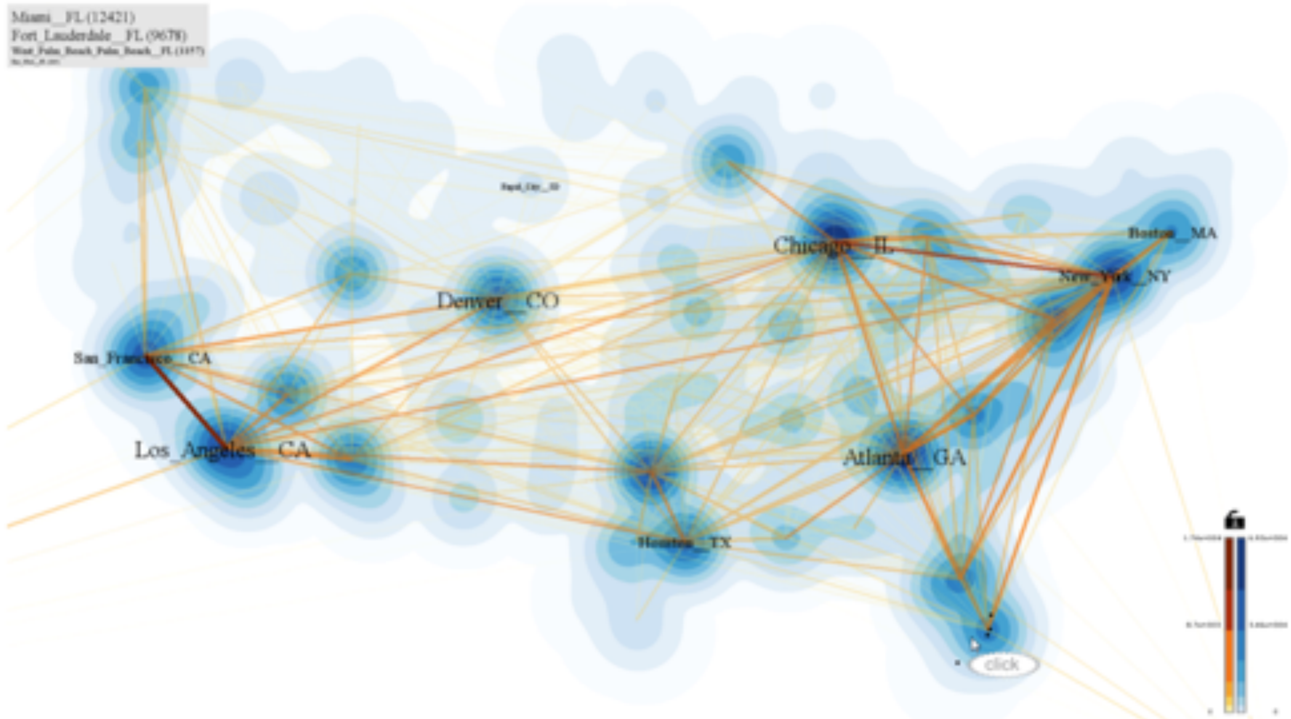
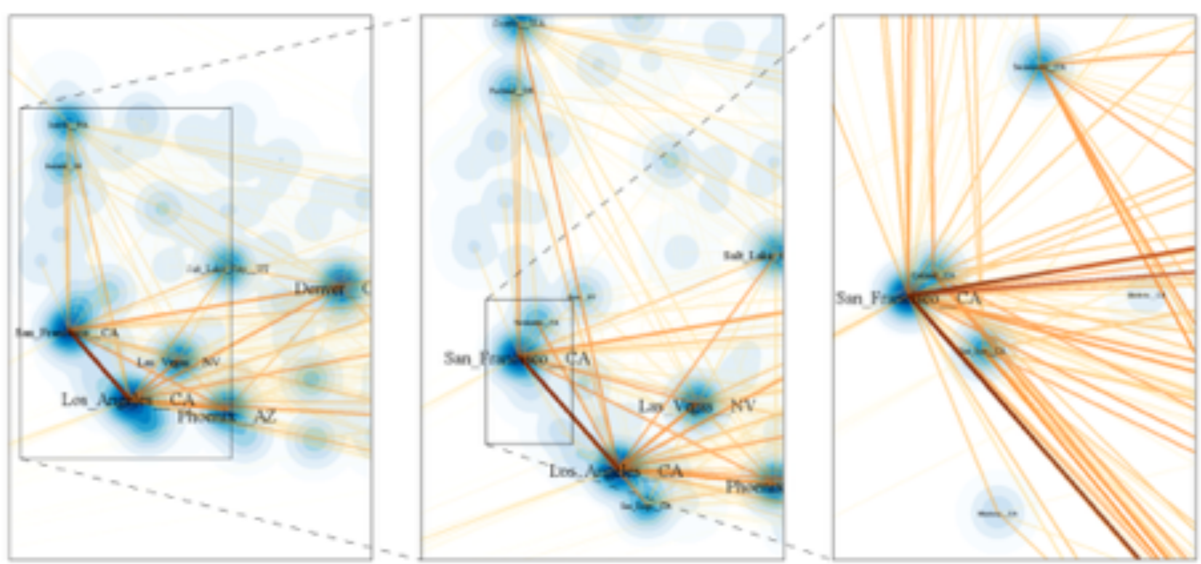
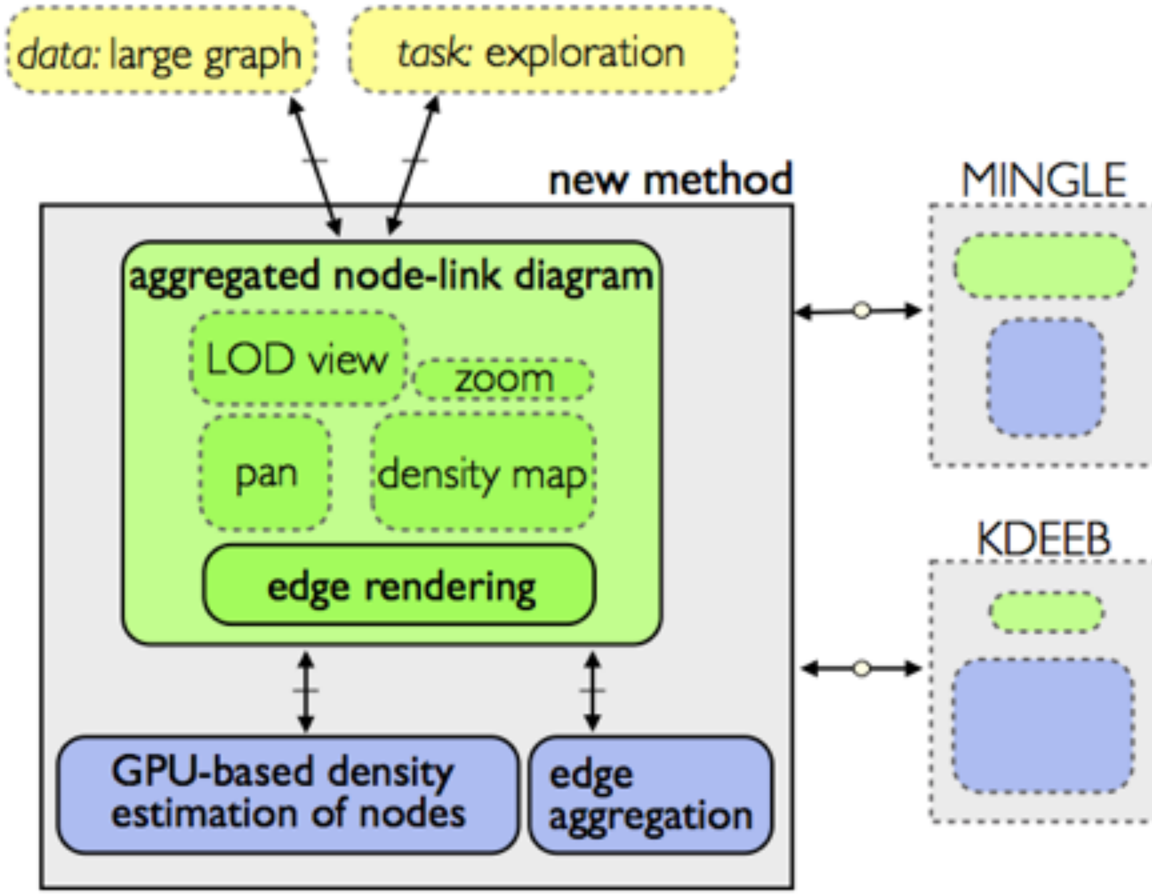


Fig. 7. US air traffic data set. The node aggregation highlights important flight hubs, while edge aggregation shows e.g. a dense connection between Los Angeles and San Francisco. A click in the Miami area (low right) highlights important nodes and a label list on the top left. From the list the user can choose interesting labels, that are placed within the visualization. The color mapping scale is shown on the bottom right.





data: large graph
task: exploration
new method
aggregated node-link diagram
LOD view
zoom
pan
density map
edge rendering
GPU-based density estimation of nodes
edge aggregation
MINGLE
KDEEB

NESTED BLOCKS AND GUIDELINES

[Meyer 2013]

implications

-comparing domains via abstractions

-generalizing techniques via abstractions

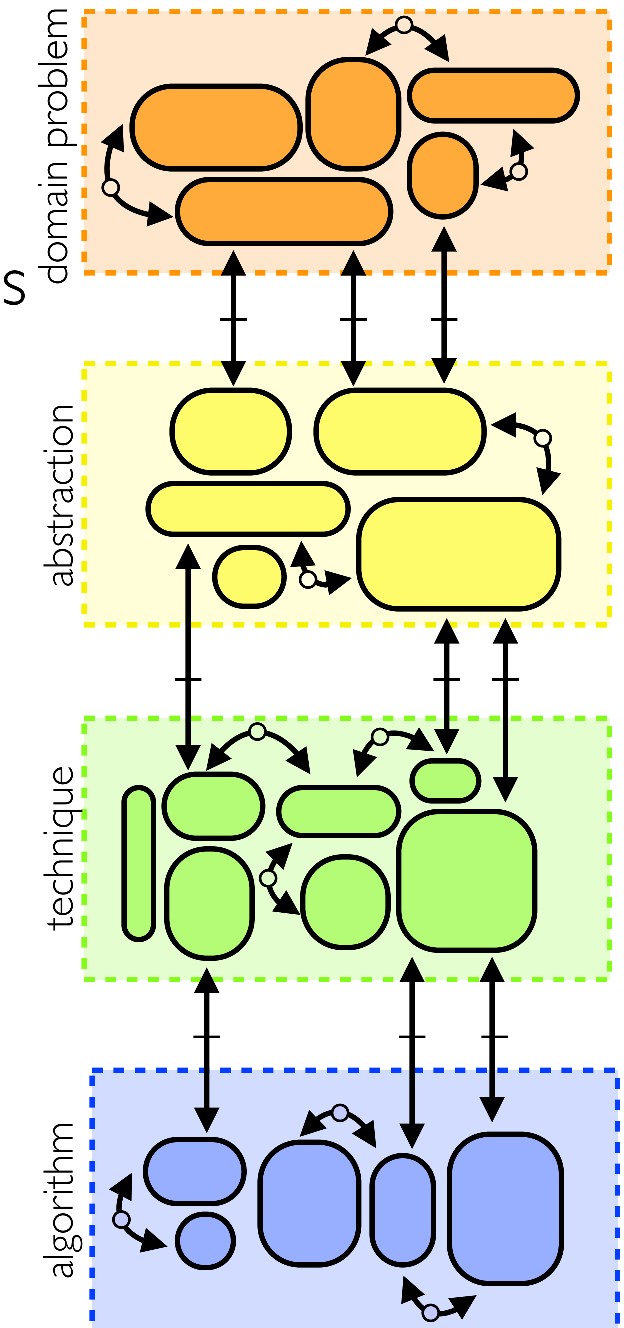
-evaluating stacks of blocks

blocks

guidelines

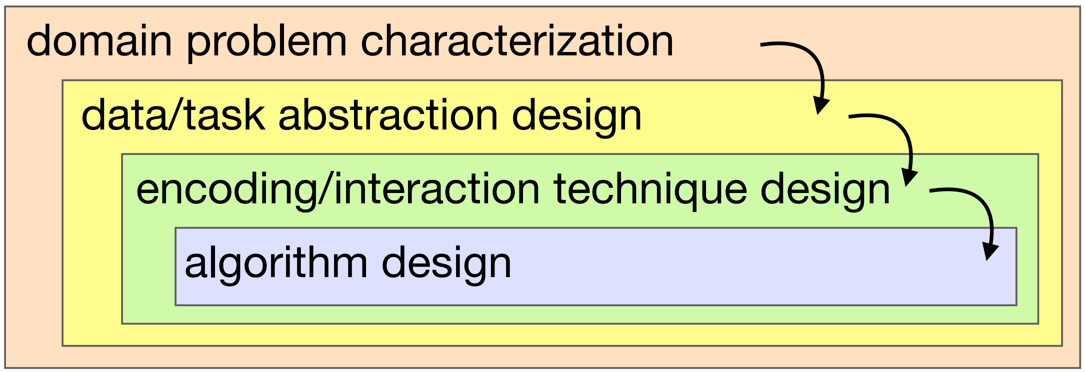
between-level guideline

within-level guideline



process models

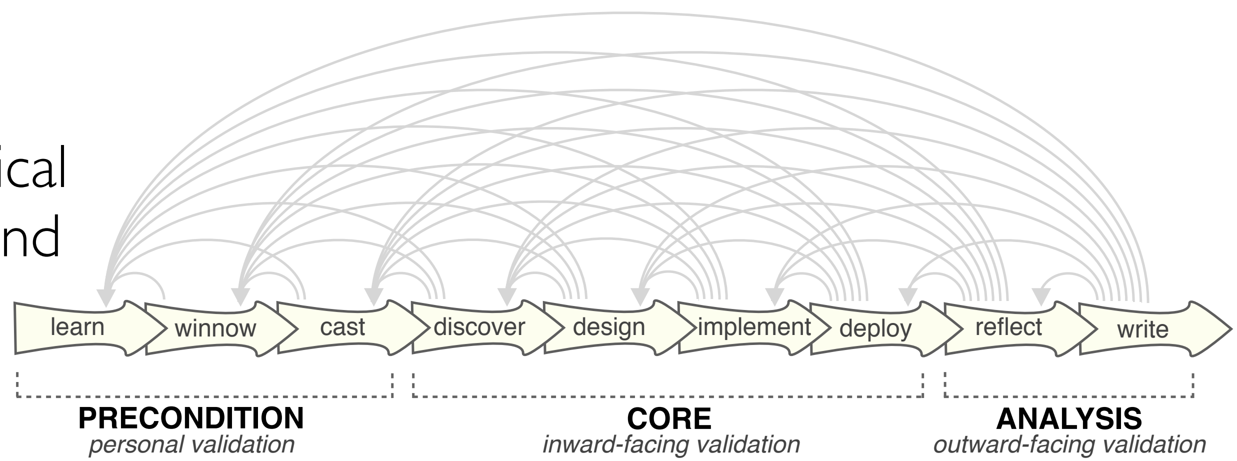
design decision models *vs* **process models**



nested model

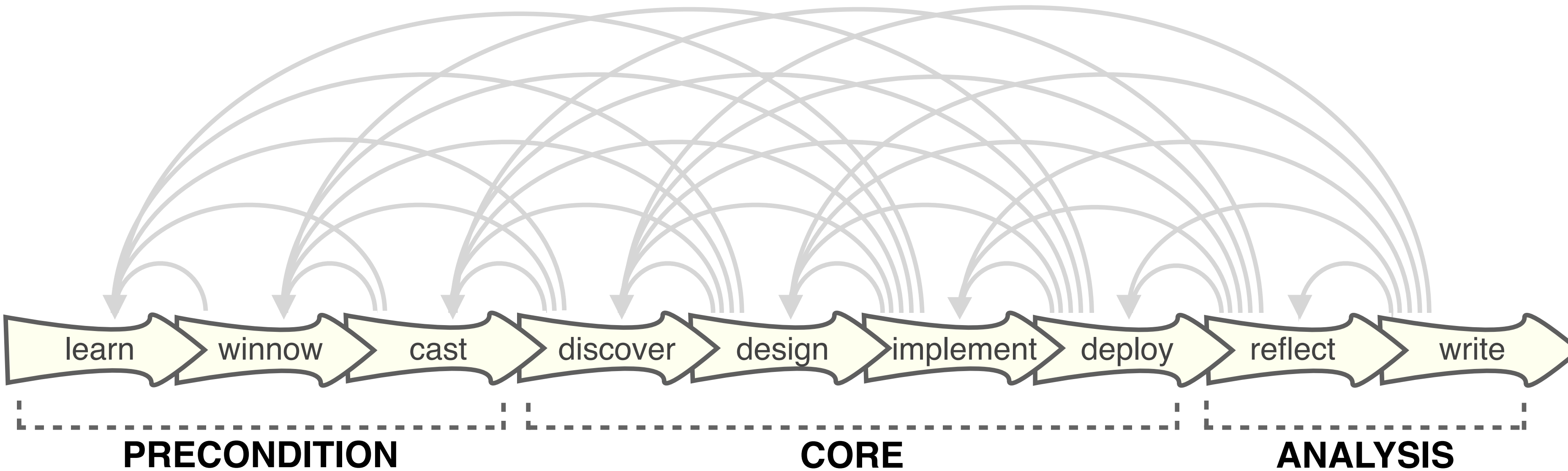
design decision model: describes levels of design inherent to, and should be considered in, the creation of a tool

process model: gives practical advice in how to design and develop a tool

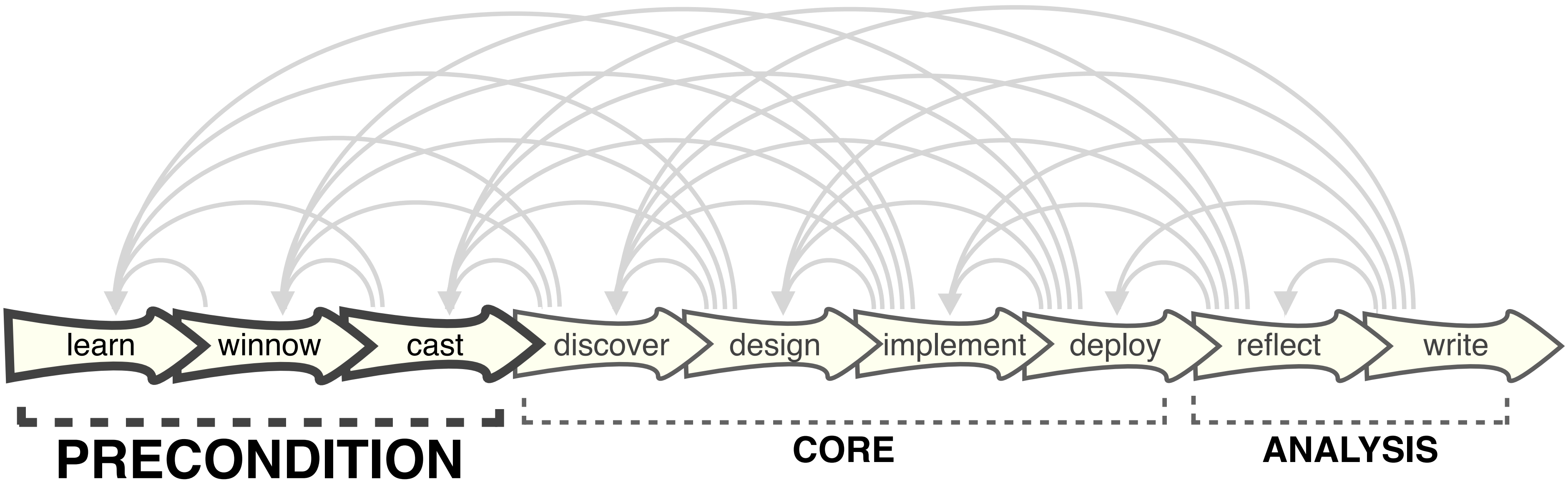


9-stage framework

the nine-stage framework

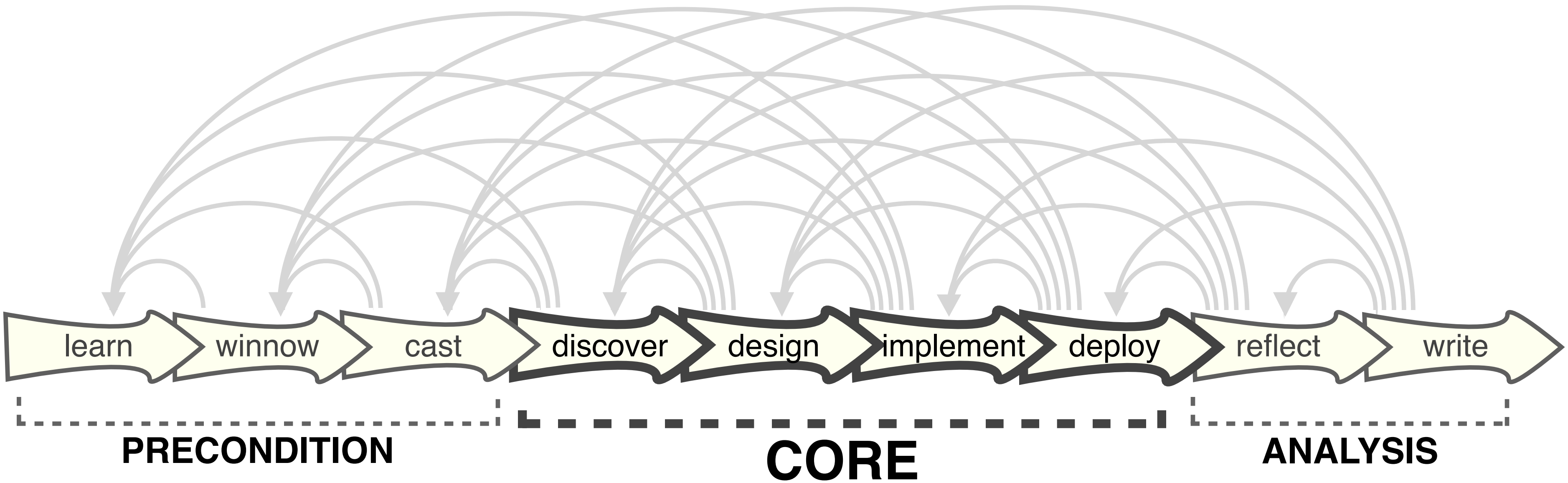


the nine-stage framework



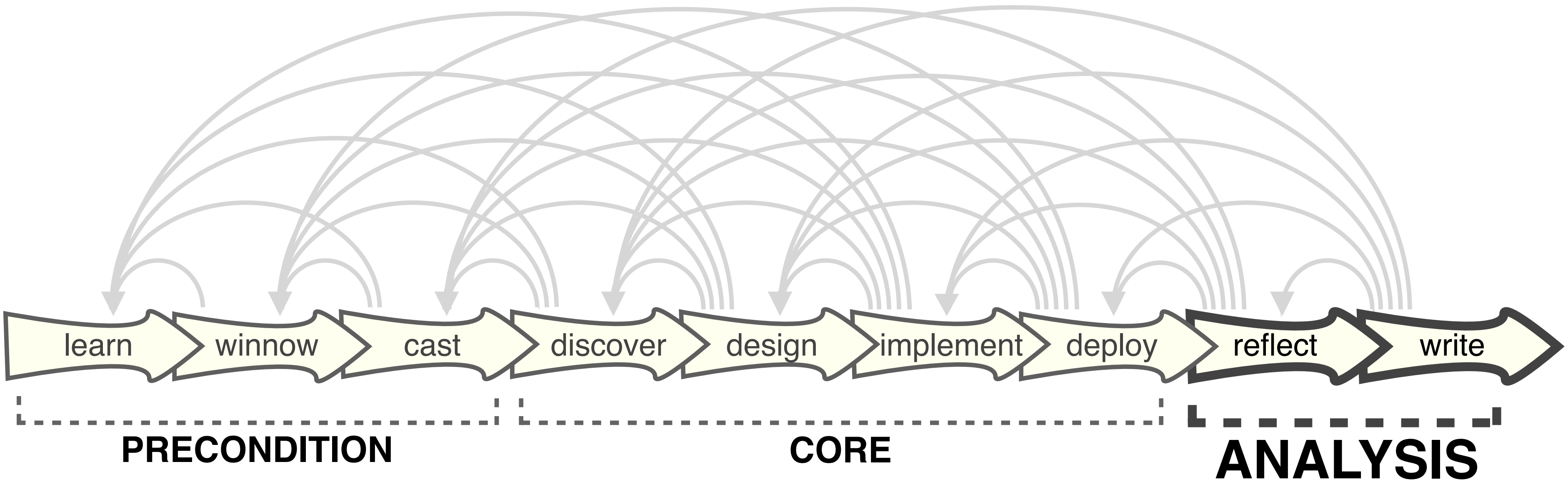
what must be done before starting a project

the nine-stage framework



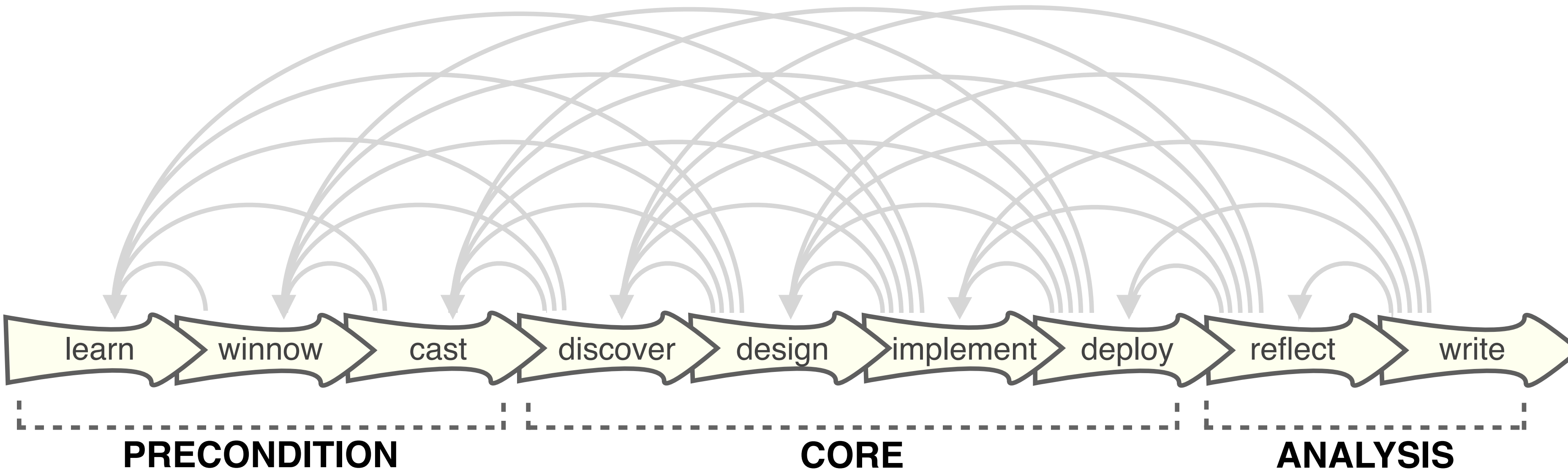
main steps of a design study

the nine-stage framework

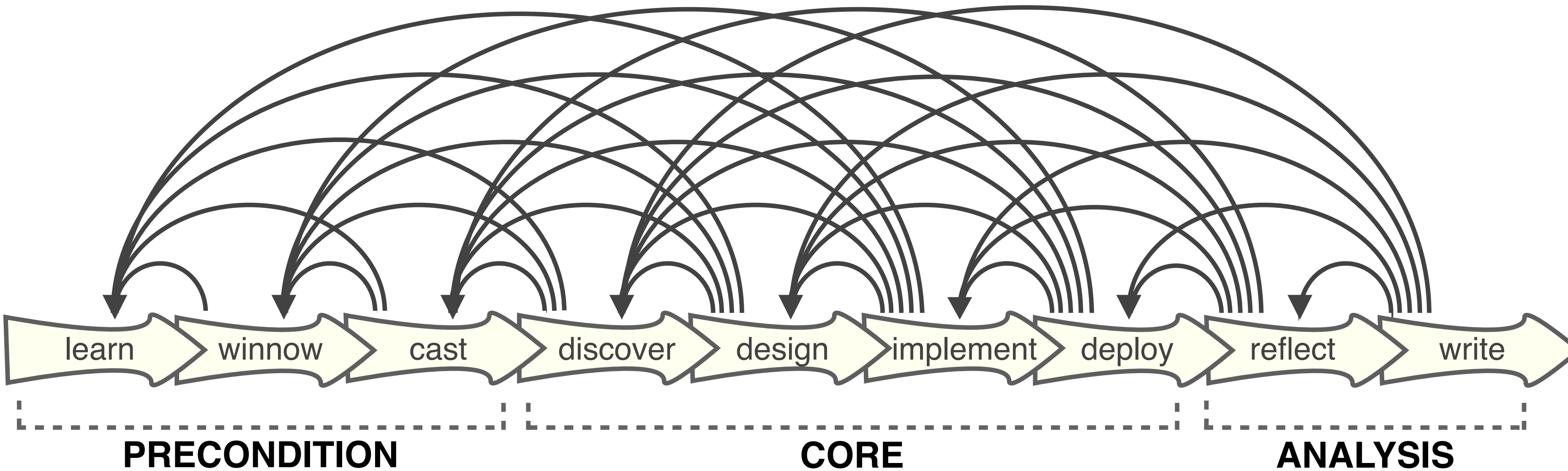


analytical reasoning at the end

the nine-stage framework



the nine-stage framework



L23: Vector and Tensor Fields

REQUIRED READING

Chapter 8

Arrange Spatial Data

8.1 The Big Picture

For datasets with spatial semantics, the usual choice for *arrange* is to *use* the given spatial information to guide the layout. In this case, the choices of *express*, *separate*, *order*, and *align* do not apply because the position channel is not available for directly encoding attributes. The two main spatial data types are geometry, where shape information is directly conveyed by spatial elements that do not necessarily have associated attributes, and spatial fields, where attributes are associated with each cell in the field. (See Figure 8.1.) For scalar fields with one attribute at each field cell, the two main visual encoding idiom families are isocontours and direct volume rendering. For both vector and tensor fields, with multiple attributes at each cell, there are four families of encoding idioms: flow glyphs that show local information, geometric approaches that compute derived geometry from a sparse set of seed points, texture approaches that use a dense set of seeds, and feature approaches where data is derived with global computations using information

Comparing 2D Vector Field Visualization Methods: A User Study

David H. Laidlaw, Robert M. Kirby, Cullen D. Jackson, J. Scott Davidson, Timothy S. Miller, Marco da Silva, William H. Warren, and Michael J. Tarr

Abstract—We present results from a user study that compared six visualization methods for two-dimensional vector data. Users performed three simple but representative tasks using visualizations from each method: 1) locating all critical points in an image, 2) identifying critical point types, and 3) advecting a particle. Visualization methods included two that used different spatial distributions of short arrow icons, two that used different distributions of integral curves, one that used wedges located to suggest flow lines, and line-integral convolution (LIC). Results show different strengths and weaknesses for each method. We found that users performed these tasks better with methods that: 1) showed the sign of vectors within the vector field, 2) visually represented integral curves, and 3) visually represented the locations of critical points. Expert user performance was not statistically different from nonexpert user performance. We used several methods to analyze the data including omnibus analysis of variance, pairwise *t*-tests, and graphical analysis using inferential confidence intervals. We concluded that using the inferential confidence intervals for displaying the overall pattern of results for each task measure and for performing subsequent pairwise comparisons of the condition means was the best method for analyzing the data in this study. These results provide quantitative support for some of the anecdotal evidence concerning visualization methods. The tasks and testing framework also provide a basis for comparing other visualization methods, for creating more effective methods and for defining additional tasks to further understand the tradeoffs among the methods. In the future, we also envision extending this work to more ambitious comparisons, such as evaluating two-dimensional vectors on two-dimensional surfaces embedded in three-dimensional space and defining analogous tasks for three-dimensional visualization methods.

Index Terms—User study, vector visualization, fluid flow visualization.

1 Introduction

One of the goals of scientific visualization is to display measurements of physical quantities so the underlying physical phenomena can be interpreted accurately, quickly,

studies help to form a basis upon which rule-of-thumb construction measures for vector visualizations can be postulated.

Design Study Methodology: Reflections from the Trenches and the Stacks

Michael Sedlmair, *Member, IEEE*, Miriah Meyer, *Member, IEEE*, and Tamara Munzner, *Member, IEEE*

Abstract—Design studies are an increasingly popular form of problem-driven visualization research, yet there is little guidance available about how to do them effectively. In this paper we reflect on our combined experience of conducting twenty-one design studies, as well as reading and reviewing many more, and on an extensive literature review of other field work methods and methodologies. Based on this foundation we provide definitions, propose a methodological framework, and provide practical guidance for conducting design studies. We define a design study as a project in which visualization researchers analyze a specific real-world problem faced by domain experts, design a visualization system that supports solving this problem, validate the design, and reflect about lessons learned in order to refine visualization design guidelines. We characterize two axes—a *task clarity axis* from fuzzy to crisp and an *information location axis* from the domain expert's head to the computer—and use these axes to reason about design study contributions, their suitability, and uniqueness from other approaches. The proposed methodological framework consists of 9 stages: *learn, winnow, cast, discover, design, implement, deploy, reflect,* and *write*. For each stage we provide practical guidance and outline potential pitfalls. We also conducted an extensive literature survey of related methodological approaches that involve a significant amount of qualitative field work, and compare design study methodology to that of ethnography, grounded theory, and action research.

Index Terms—Design study, methodology, visualization, framework.

1 Introduction

Over the last decade design studies have become an increasingly popular approach for conducting problem-driven visualization research. Design study papers are explicitly welcomed at several visualization venues as a way to explore the choices made when applying visualization techniques to a particular application area [55], and many exemplary design studies now exist [17, 34, 35, 56, 94]. A careful reading of these papers reveals multiple steps in the process of conducting a design study, including analyzing the problem, abstracting data and tasks, designing and implementing a visualization solution, evaluating the solution with real users, and writing up the findings.

And yet there is a lack of specific guidance in the visualization literature that describes holistic methodological approaches for conducting design studies—currently only three paragraphs exist [49, 55]. The relevant literature instead focuses on methods for designing [1, 42, 66, 79, 82, 90, 91] and evaluating [13, 33, 39, 50, 68, 69, 76, 80, 85, 86, 95] visualization tools. We distinguish between methods and methodology with the analogy of cooking; *methods* are like ingredients, whereas *methodology* is like a recipe. More formally, we use Crotty's definitions that methods are "techniques or procedures" and a methodology is the "strategy, plan of action, process, or design lying behind the choice and use of particular methods" [18].

From our personal experience we know that the process of conducting a design study is hard to do well and contains many potential pitfalls. We make this statement after reflecting on our own design studies, in total 21 between the 3 authors, and our experiences of reviewing many more design study papers. We consider at least 3 of our own design study attempts to be failures [51, 54, 72]; the other 18 were more successful [4, 5, 10, 40, 43, 44, 45, 46, 52, 53, 67, 70, 71,

of visualization a good idea at all? How should we go about collaborating with experts from other domains? What are pitfalls to avoid? How and when should we write a design study paper? These questions motivated and guided our methodological work and we present a set of answers in this paper.

We conducted an extensive literature review in the fields of human computer interaction (HCI) [7, 8, 9, 12, 16, 19, 20, 21, 22, 25, 26, 27, 28, 29, 30, 31, 38, 47, 57, 63, 64, 65, 83] and social science [6, 14, 18, 24, 32, 62, 81, 87, 93] in hopes of finding methodologies that we could apply directly to design study research. Instead, we found an intellectual territory full of quagmires where the very issues we ourselves struggled with were active subjects of nuanced debate. We did not find any off-the-shelf answers that we consider suitable for wholesale assimilation; after careful gleaning we have synthesized a framing of how the concerns of visualization design studies both align with and differ from several other qualitative approaches.

This paper is the result of a careful analysis of both our experiences in the "trenches" while doing our own work, and our foray into the library "stacks" to investigate the ideas of others. We provide, for the first time, a discussion about design study methodology, including a clear definition of design studies as well as practical guidance for conducting them effectively. We articulate two axes, *task clarity* and *information location*, to reason about what contributions design studies can make, when they are an appropriate research device, and how they are unique from other approaches. For practical guidance we propose a process for conducting design studies, called the *nine-stage framework*, consisting of the following stages: *learn, winnow, cast, discover, design, implement, deploy, reflect,* and *write*. At each stage