Designing With Diagrams

Chris Martens
Computer Science Department
Carnegie Mellon University

Human Aspects of Software Development
Spring 2011

Under scrutiny:

How Software Engineers *plan and design* software using diagrams/visual notations.

What's a Diagram?

a simplified and structured visual representation that shows entities and relationships representing the architecture or implementation of a software system.

[Cherubini et. al. 2007 - "Let's Go to the Whiteboard"]

Not:
Visual Programming
Code Visualization

Design diagrams come into play *separately from the act of programming.*

A good first question:

Are (visual) metaphors actually useful?

"[...] software engineers frequently talk about software behavior in terms of what a component 'knows' or is 'trying to do.' This strikes many as sloppy and imprecise, hence undesirable. Dijkstra [12] has gone so far as to suggest that computer science faculty implement a system of fines to stamp it out among their students, although he acknowledges this would be very difficult to do."

[Herbsleb 1999 - "Metaphorical Representation in Collaborative Software Engineering"]

Several papers bring up:

- The efficacy of drawings in other disciplines (architecture, mechanical and electrical engineering -- obvious *spatial metaphors*)
- Cognitive science literature supporting ease of mental processing: *parallel* instead of sequential; compact instead of linear.

But maybe SE is different?

Before we can make such judgments...

"Let's Go to the Whiteboard: How and Why Software Developers Use Drawings"

[Cherubini et. al. 2007]

Four central concerns:

A. How do engineers use diagrams in their work?

B. Why do engineers use diagrams in their work?

C. What graphical conventions do engineers use?

D. What is the culture around these drawings?

Their method:

Interviews and surveys of Microsoft developers.

- 45 minute semi-structured interview: *what, when, why, how* questions
- Broader survey on nine recurring scenarios

Findings

Visual conventions:

- Iconic representations (db = cylinder, computer = tower, person = stick figure)
- Circles (state diagrams) and boxes
- Labels; meaningful size
- Arrows for relationships: usually pointing rightward or downward
- Colors rarely used

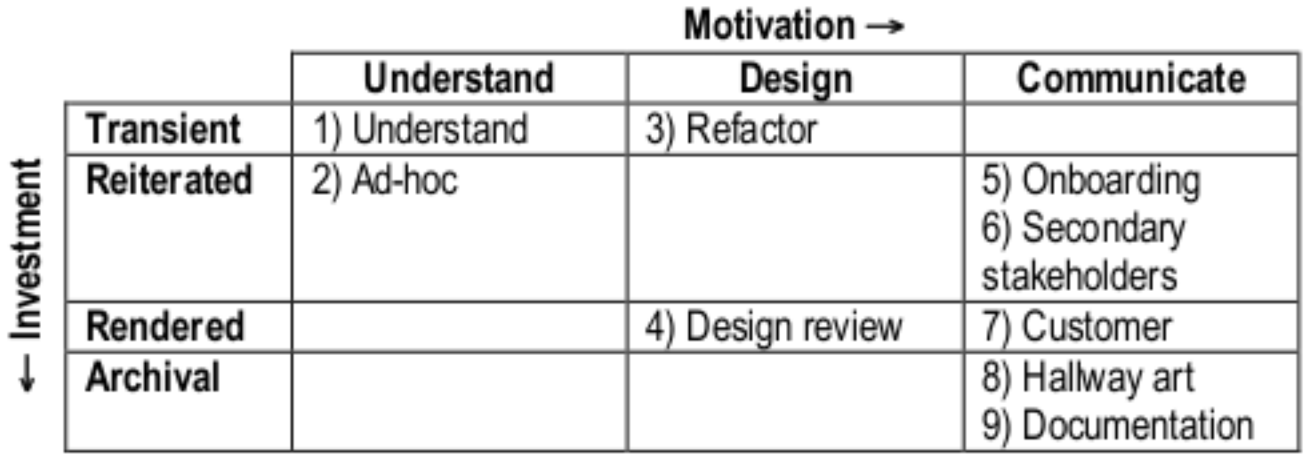
Findings

Many entities and relationships:

- Classes, methods, binaries, processes, databases, hardware, UI screens, states, people
- Inheritance, data reference (e.g. pointers), data access ("talks to"), procedure call, message passing, transition, containment

Findings

Motivations and Scenarios:



Motivation →

↓ Investment

	Understand	Design	Communicate
Transient	1) Understand	3) Refactor	
Reiterated	2) Ad-hoc		5) Onboarding 6) Secondary stakeholders
Rendered		4) Design review	7) Customer
Archival			8) Hallway art 9) Documentation

Table 2: The model of diagram use derived from interviews and survey responses. Scenarios are categorized by the developer's motivation for creating the drawing and the developer's investment in the evolution process of the drawing.

Answers to questions

A. How do engineers use diagrams in their work?

- Transient forms for exploration (whiteboards, scrap paper, notebooks
- More permanent renderings for communication with others

"The transience of casual sketches seems to be a difference with other disciplines like architecture, where these are often archived with great care as a record of design process."

Answers to questions

C. Which graphical conventions are used?

- Solo/peer-to-peer: no graphical standard! (like UML...) Assumed meanings depend on context.
- Visualizations in documentation out-of-date; rarely used.

Answers to questions

D. What is the culture around these drawings?

- Limited adoption of drawing tools
- Developers remain focused on the code itself.
- Particular diagrams had value; used for design reiteration.
- Diagrams generated automatically "less interesting" than those produced in a collaborative effort

Tool Implications?

"Many of the developers interviewed suggested that they desired some sort of 'intelligent whiteboard' to augment the drawing process and capture the result in electronic form."

So they made a prototype...

Building an Ecologically valid, Large-scale Diagram to Help Developers Stay Oriented in Their Code

[Cherubini et. al. 2007]

The "code map"

A large paper "map" of the types and relationships in the existing code base, updated every day, hung up in the hallway.

After initial feedback: also "MiniMaps".

But no one used it

Very little interaction, except for onboarding.

Why?

- Wrong level of detail
- Content too static: couldn't display different relationships for different use cases
- Layout too static: couldn't zoom or rearrange to adapt to different conversation scope.

A different angle

The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering

[Moody 2009]

A different angle

Along similar lines to "cognitive dimensions", applied specifically to diagrams.

[...but, they criticize, CD's "level of generality precludes specific predictions, meaning that it is unfalsifiable."]

Objectives

To "establish the foundations for a science of visual notation design."

Points out that SE has neither theory nor a body of empirical evidence for design practices -- leads to repeating common practices without careful thought.

Ontological analysis/mappings

There should be a one-to-one mapping between notation (visual constructs) and ontology (formal concepts).

Violations: construct deficit, overload, redundancy, excess

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
4.5 Cognitive Integration
4.6 Visual Expressiveness
4.7 Dual Coding
4.8 Graphic Economy
4.9 Cognitive Fit

Principles:

4.1 Semiotic Clarity

1:1 correspondence between semantic constructs and graphical symbols.

Principles:

4.1 Semiotic Clarity

4.2 Perceptual Discriminability

Different symbols should be distinguishable.

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
Use representations whose appearance suggests their meaning.

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
Include explicit mechanisms for dealing with complexity (e.g. modularization/hierarchical structure)

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
4.5 Cognitive Integration
Include explicit mechanisms to support integration of information from different diagrams (e.g. contextualization; navigation)

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
4.5 Cognitive Integration
4.6 Visual Expressiveness
Use the full range of visual variables (not just shape... texture, brightness, size, color)

Principles:

- 4.1 Semiotic Clarity
- 4.2 Perceptual Discriminability
- 4.3 Semantic Transparency
- 4.4 Complexity Management
- 4.5 Cognitive Integration
- 4.6 Visual Expressiveness
- **4.7 Dual Coding**
  - Use text to complement graphics

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
4.5 Cognitive Integration
4.6 Visual Expressiveness
4.7 Dual Coding
4.8 Graphic Economy
The number of different symbols should be cognitively manageable

Principles:

4.1 Semiotic Clarity
4.2 Perceptual Discriminability
4.3 Semantic Transparency
4.4 Complexity Management
4.5 Cognitive Integration
4.6 Visual Expressiveness
4.7 Dual Coding
4.8 Graphic Economy
4.9 Cognitive Fit
Different visual dialects for different audiences

Empirical Validation?

Comprehension of diagram syntax: an empirical study of Entity Relationship notations

[Purchase et. al. 2004]

Aims:

- Raise the issue of criteria by which notation may be chosen
- Propose a methodology to compare two complete notations
- Demonstrate an application of the methodology by performing an experiment

Two forms of Entitity Relationship (ER) Diagrams

Chen

vs

SSADM

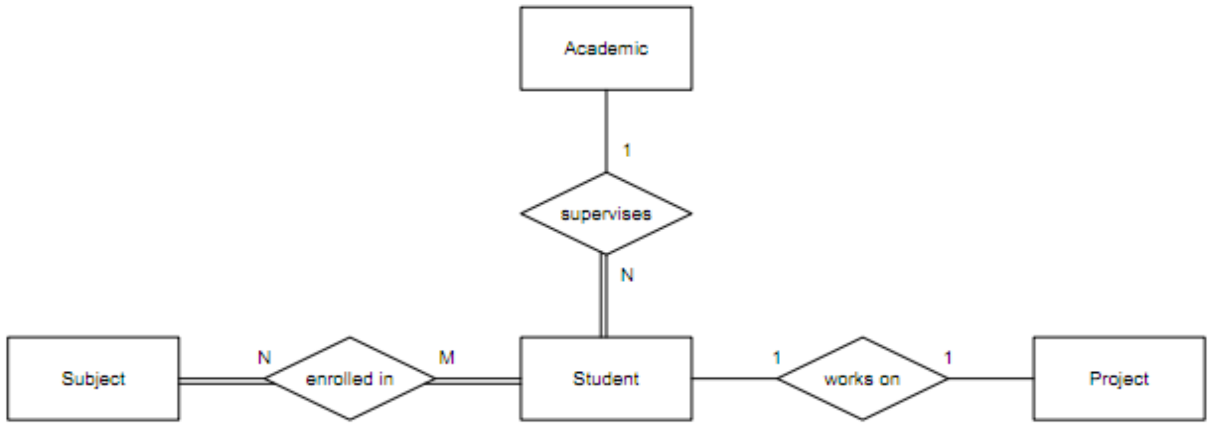


Figure 1: The student application using the Chen notation [1]

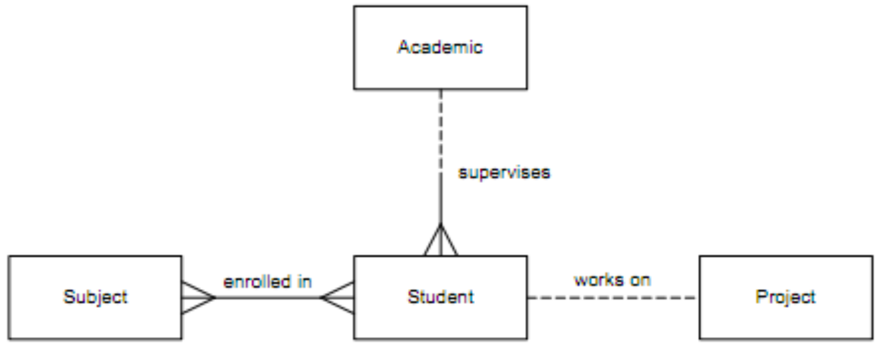


Figure 2: The student application using the SSADM notation [2]

Experimental setup

Participants (university students) given a tutorial on the two notations, then a textual specification plus several diagrams: asked to indicate (yes/no) whether the diagrams meets the spec.

Measurements: correctness and response time.

Findings

SSADM notation better understood.

Faster response times, but error rate the same.

Subjectively preferred (particularly cardinality notation)

Primary conclusion: "conciseness" wins.

Conclusion

IMO:

A lot of this work seems inconclusive.

To what extent can we apply design principles and results of studies to the ad-hoc diagram style commonly used?

To what extent are SE diagrams helpful *when compared to* text?