Effectiveness Sans Formality

And

Foundations Of Software Development

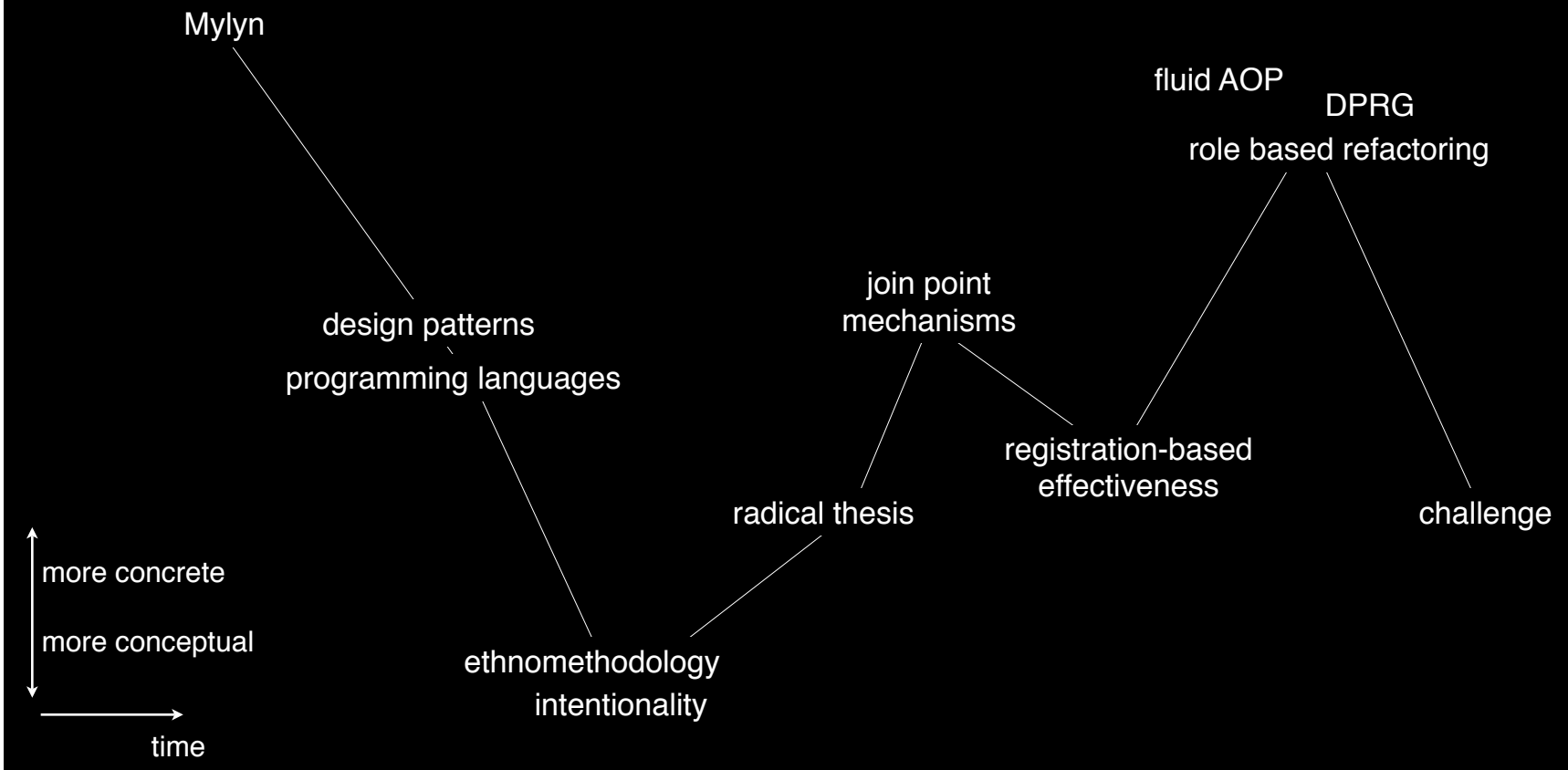
Gregor Kiczales
University of British Columbia

© Gregor Kiczales, 2007. Some rights reserved.
Except where otherwise indicated, this work is licensed under
http://creativecommons.org/licenses/by-sa/2.5/ca/

Topics

- Two conflicts
  - programming languages vs. design patterns, Mylyn
    - PL power vs. flexibility w/ 'high-level' abstractions
  - our conceptions of field vs. work of PARC colleagues
    - proper role of formality in foundations of software?
- Can a single 'account' resolve these?
- Can it clarify and support current work?
- Can it point to improved concepts and techniques?

Timeline


Mylyn
fluid AOP
DPRG
role based refactoring
join point
mechanisms
design patterns
programming languages
registration-based
effectiveness
radical thesis
challenge
more concrete
more conceptual
ethnomethodology
intentionality
time

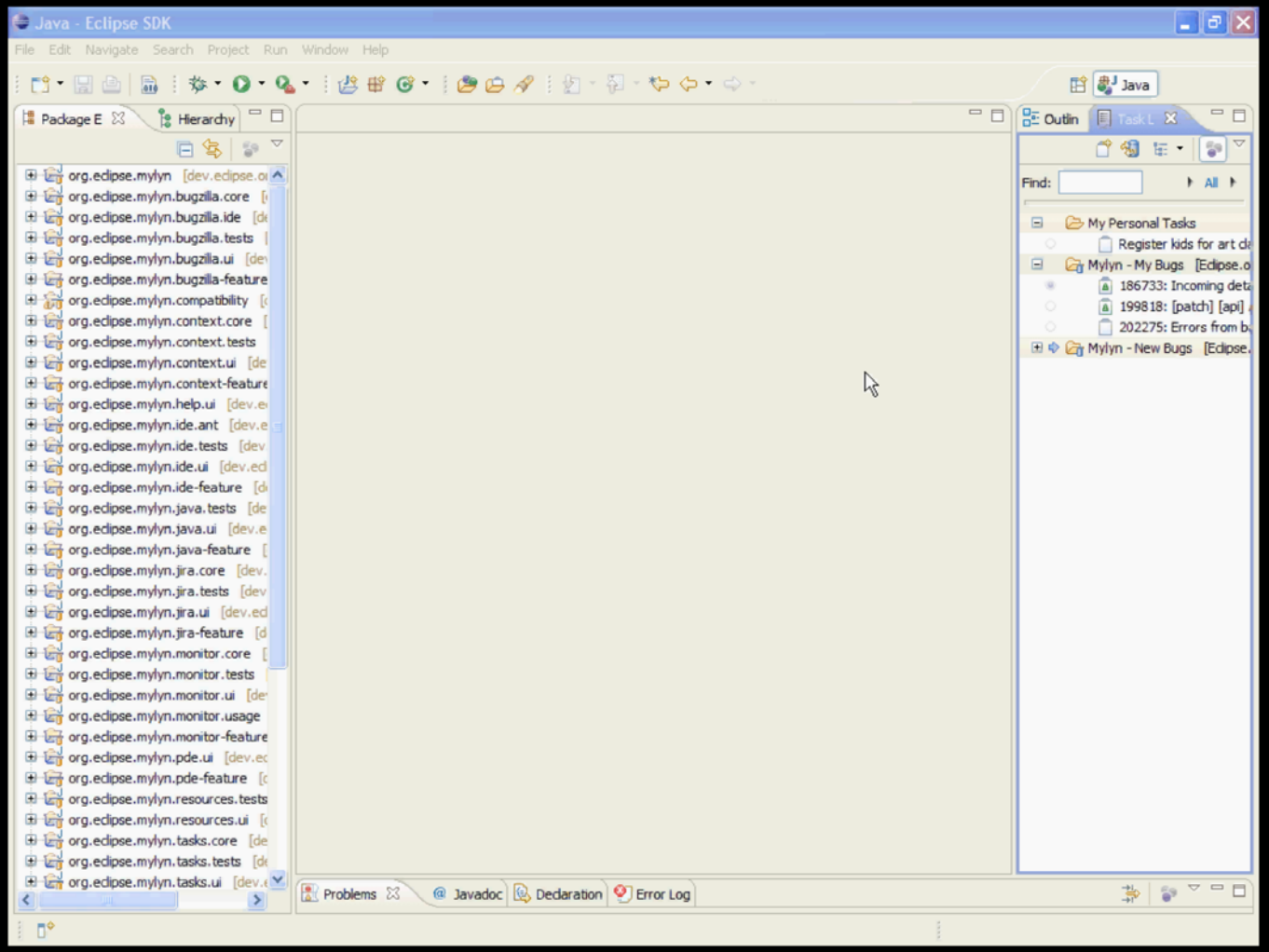
Tactics

- Goal is to draw connections and possible directions
  - breadth of topics
  - for each some of you will be more expert
  - trying to be informed; but will surely make mistakes
- Will use work I know best as examples
  - Mylyn, AOP, Fluid AOP, DPRG, role-based refactoring
  - not claiming that these are the best examples
  - just the ones I know best

Mylyn (née Mylar)

[Kersten, Murphy et. al., www.eclipse.org/mylyn]

Mylyn (née Mylar)



[Kersten, Murphy et. al., www.eclipse.org/mylyn]

Mylyn (née Mylar)

- Definition of task context (a concern)
  - not declared explicitly, not formalized
  - evolves from watching developer(s)
  - not rigidly enforced, adaptive
  - socially constructed and negotiated
- Lightweight management of higher-level structures
  - emergent, crosscutting...
- Very fast adoption

Debate on PL and Tool Support for Design Patterns

Vlissides writes:

As we worked on our patterns, it wasn't long before we began to think about automating their application. By mid-1995 we had developed a tool for browsing patterns on-line and for generating their implementations automatically [5]. It taught us much about the relative merits of patterns and tools.

...

As you master the patterns, however, the drawbacks of the tool become apparent. Its main weakness is inflexibility. Developers use patterns in surprising ways, often as starting points from which to evolve specialized solutions that the pattern author(s) never foresaw. The code generator can't help you there. In fact, it makes things worse, in three ways:

1. It's hard to generate code that's as flexible as the pattern is. The generator can be designed to cover the trade-offs and variants that are explicit in the pattern, but it can't vary far from them.

...

[Chambers, Harrison, Vlissides, POPL 2000]

Debate on PL and Tool Support for Design Patterns

2. Generated code is often difficult to integrate with existing code, especially when they're in a language that lacks multiple inheritance. More advanced tooling (e.g., subject-oriented support) might alleviate this problem.

3. Whenever you generate code, you pave the way for the so-called round-trip problem: Unless you're careful, regenerating the code will overwrite modifications to the previous generation. Because most pattern implementations do not involve a lot code, common solutions to this problem [26] render the whole approach more trouble than it's worth. The most useful aspect of the tool turned out to be decidedly low tech: the book text itself in HTML form, with hyperlinks for all cross-references. It made navigating and searching the text much easier, not to mention saving you from lugging the book around if you already carry a notebook computer.

[Ibid.]

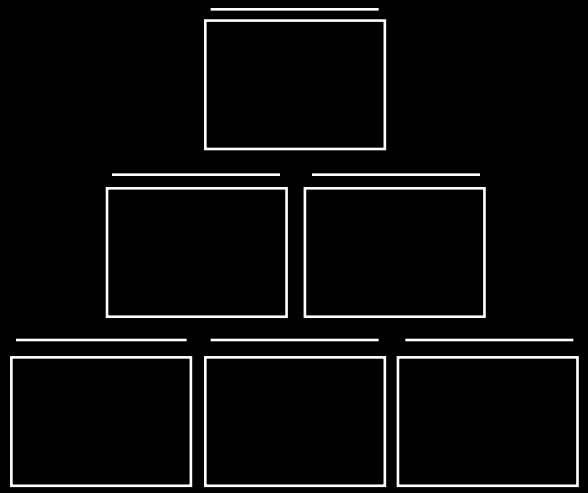
Design Patterns and Formality

- Formalized patterns too inflexible in the face of
  - surprising uses
  - surprising combinations
  - integration w/ situation particulars
- Meaning, applicability, form are socially mediated
- Hyper-linked documentation most useful
- Very fast adoption

Programming Languages

- Programs are effective formal abstractions of computation
  - effective: programs produce computations (¬ 'just' models)
  - formal: formal system, crisp, discrete...
  - abstraction: of the computation
- Effective composition of abstraction
  - well-defined, context insensitive, orthogonal
  - OO, declarative, functional...
  - supports reasoning from small set of principles

‘Effective Formality All The Way Down’



- Languages, layers, APIs, components, frameworks, DSLs...
- Each interface presents effective abstraction
- Each module fully implements higher level
- ‘Effective formality all the way down’
  - from highest levels
  - down to the machine code
  - everything is in formal effective notation

PARC Context

- Lucy Suchman et. al.
  - ethnomethodological studies of work...
  - Plans and Situated Actions
- Brian Smith
  - foundations of computation, intentionality
  - On the Origin of Objects
- Perceived common threads:
  - the world isn't formal; at least not 'all the way down'
  - that has real consequences for software

Plans & Situated Actions

- Lucy Suchman's Ph.D. dissertation
  - expanded in 2007: Human-Machine Reconfigurations
- Many things, including:
  - a discussion of 'smart' user-interface design
  - a response to planning style AI of early 80s
  - how people work together
    - construct shared understanding
    - roles played by artifacts

The notion that we act in response to an objectively given social world is replaced by the assumption that our everyday social practices render the world publicly available and mutually intelligible. It is those practices that constitute ethnomethods. The methodology of interest to ethnomethodologists, in other words, is not their own but that deployed by members of the society in coming to know, and making sense out of, the everyday world of talk and action.

[Suchman, *Human-Machine Reconfigurations*: 76]

The ethnomethodological view of purposeful action and shared understanding is outlined in this chapter under five propositions:

1. Plans are representations of situated actions;
2. In the course of situated action, representation occurs when otherwise transparent activity becomes in some way problematic;
3. The objectivity of the situations of our action is achieved rather than given;
4. A central resource for achieving the objectivity of situations is language, which stands in a generally indexical relationship to the circumstances that it presupposes, produces and describes;
5. As a consequence of the indexicality of language, mutual intelligibility is achieved on each occasion of interaction with reference to situation particulars rather than being discharged once and for all by a stable body of shared meanings.

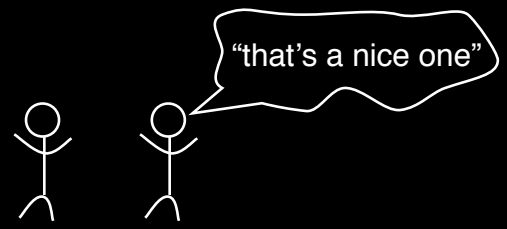
[Suchman, *Human-Machine Reconfigurations*: 70]

Expressions that rely on their situation for significance are commonly called *indexical*, after the "indexes" of Charles Peirce (1933), the exemplary indexicals being first- and second-person pronouns, tense and specific time and place adverbs such as *here* and *now.*

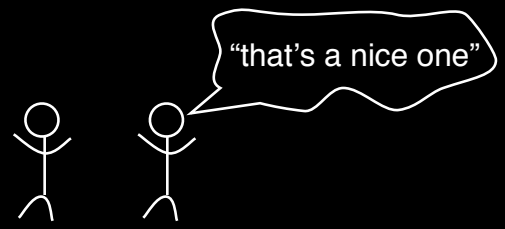
[Ibid.: 78]

Expressions that rely on their situation for significance are commonly called *indexical*, after the "indexes" of Charles Peirce (1933), the exemplary indexicals being first- and second-person pronouns, tense and specific time and place adverbs such as *here* and *now.*

[Ibid.: 78]



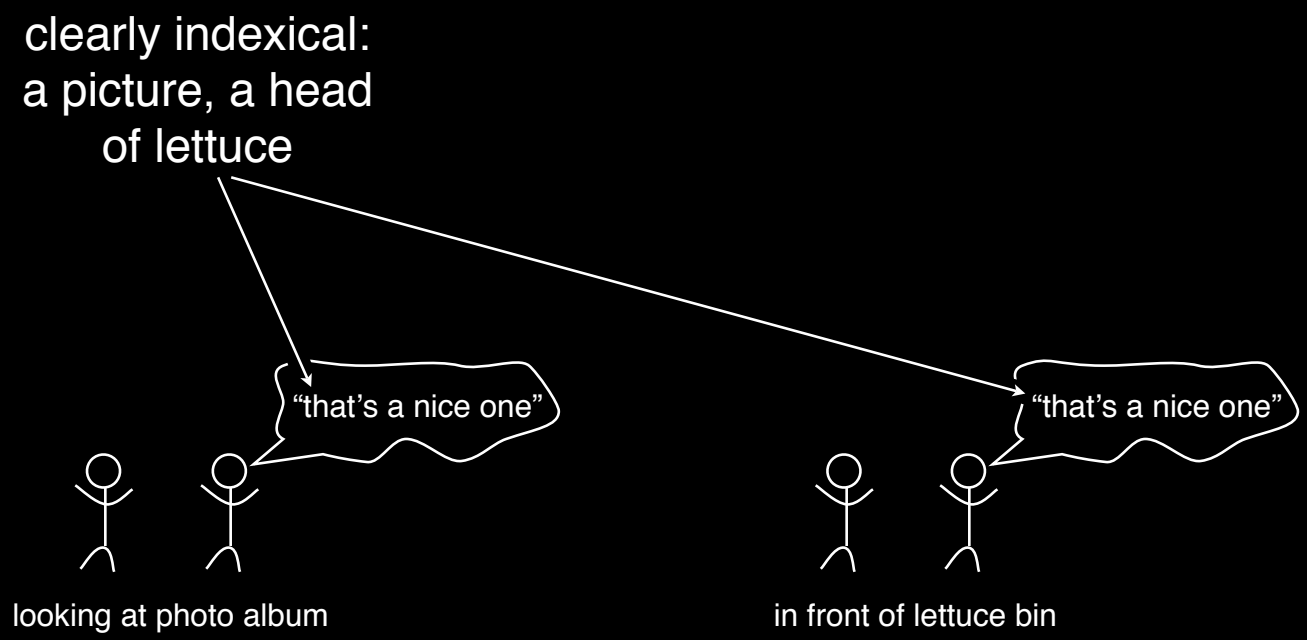
looking at photo album



in front of lettuce bin

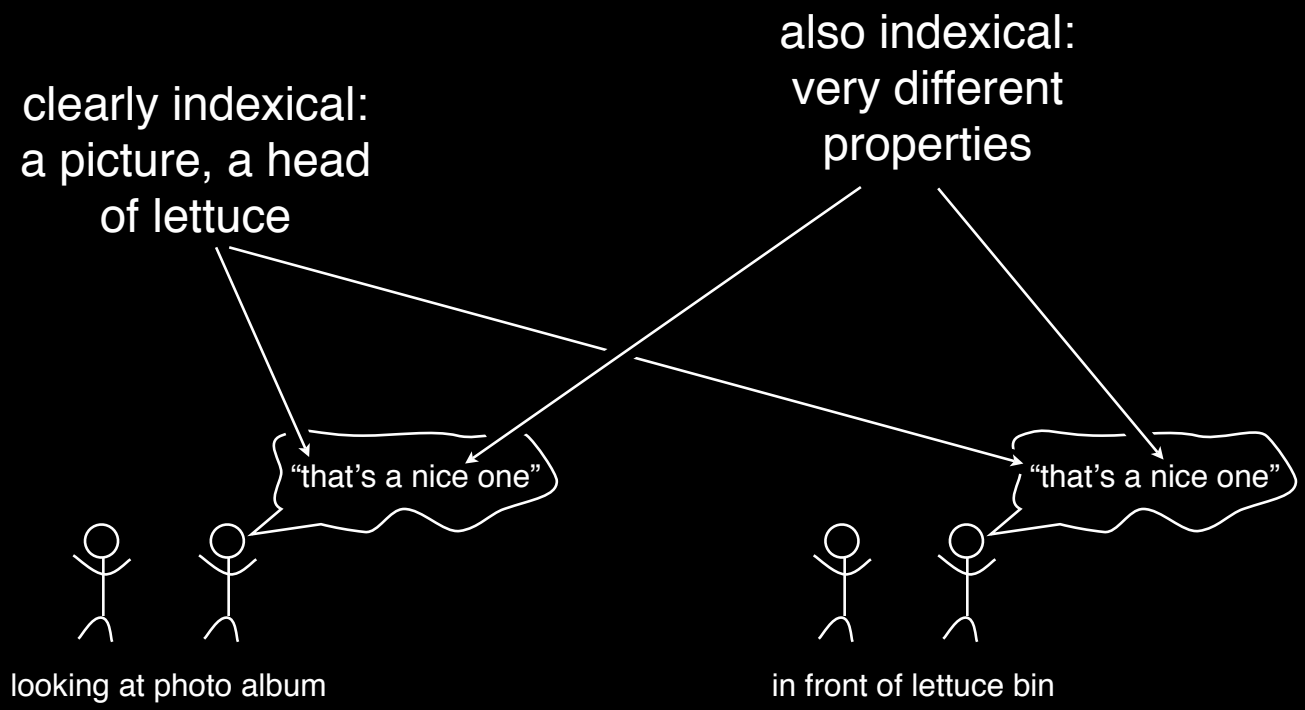
Expressions that rely on their situation for significance are commonly called *indexical*, after the "indexes" of Charles Peirce (1933), the exemplary indexicals being first- and second-person pronouns, tense and specific time and place adverbs such as *here* and *now.*

[Ibid.: 78]



Expressions that rely on their situation for significance are commonly called *indexical*, after the "indexes" of Charles Peirce (1933), the exemplary indexicals being first- and second-person pronouns, tense and specific time and place adverbs such as *here* and *now.*

[Ibid.: 78]



...visitor and host will never establish in just so many words precisely what it is that the visitor intends and the host understands. Their interpretation of the term [nice] will remain partially unarticulated, located in their unique relationship to the photograph and to the context of the remark. Yet the shared understanding they do achieve will be perfectly adequate for purposes of their interaction. [Ibid.: 78]

"[S]peakers can...do the immense work they do with natural language, even though over the course of their talk it is not known, and is never, not even "in the end," available for saying in so many words just what they are talking about. Emphatically, that does not mean that speakers do not know what they are talking about, *but instead that they know what they are talking about in that way.*" [Garfield and Sacks 1970: 342-4, original emphasis]

In this sense deictic expressions, time and place adverbs and pronouns are just particularly clear illustrations of the general fact that all language, including the most abstract or eternal, stands in an essentially indexical relationship to the embedding world.

[Suchman, *Human-Machine Reconfigurations*: 79]

On the Origin of Objects

- Brian Cantwell Smith, 1996
- Started as project in foundations of computation
  - became exploration of intentionality and ontology
  - "a new metaphysics--a philosophy of presence"
- How does intentionally work, arise...

Intentionality is the power of minds to be about, to represent, or to stand for, things, properties and states of affairs. The puzzles of intentionality lie at the interface between the philosophy of mind and the philosophy of language.[1]

This is different than intention and intent:

A course of action that one intends to follow.[2]

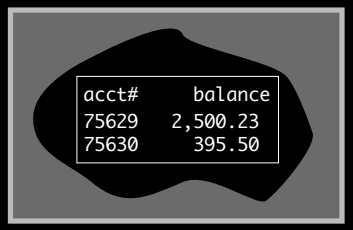
And different than intension and intensionality:

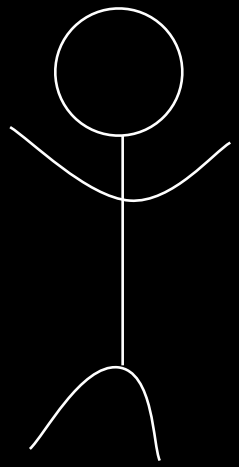
Any property or quality connoted by a word, phrase or other symbol. Intension is generally discussed with regard to extension (or denotation). Intension refers to the set of all possible things a word or phrase could describe, extension to the set of all actual things the word describes.[3]

---

1. Stanford Encyclopedia of Philosophy.
2. American Heritage Dictionary.
3. Wikipedia.

Developing A Trivial Banking System



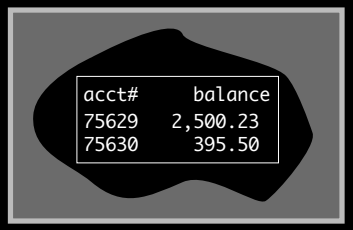


```
void deposit(Cash x) {
  ...
}
```

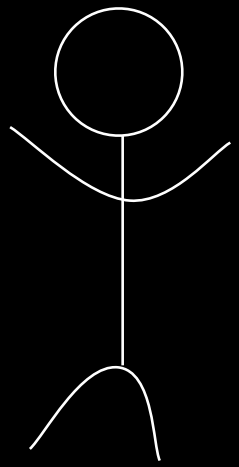
<Sue's account>

Developing A Trivial Banking System

concrete



concrete



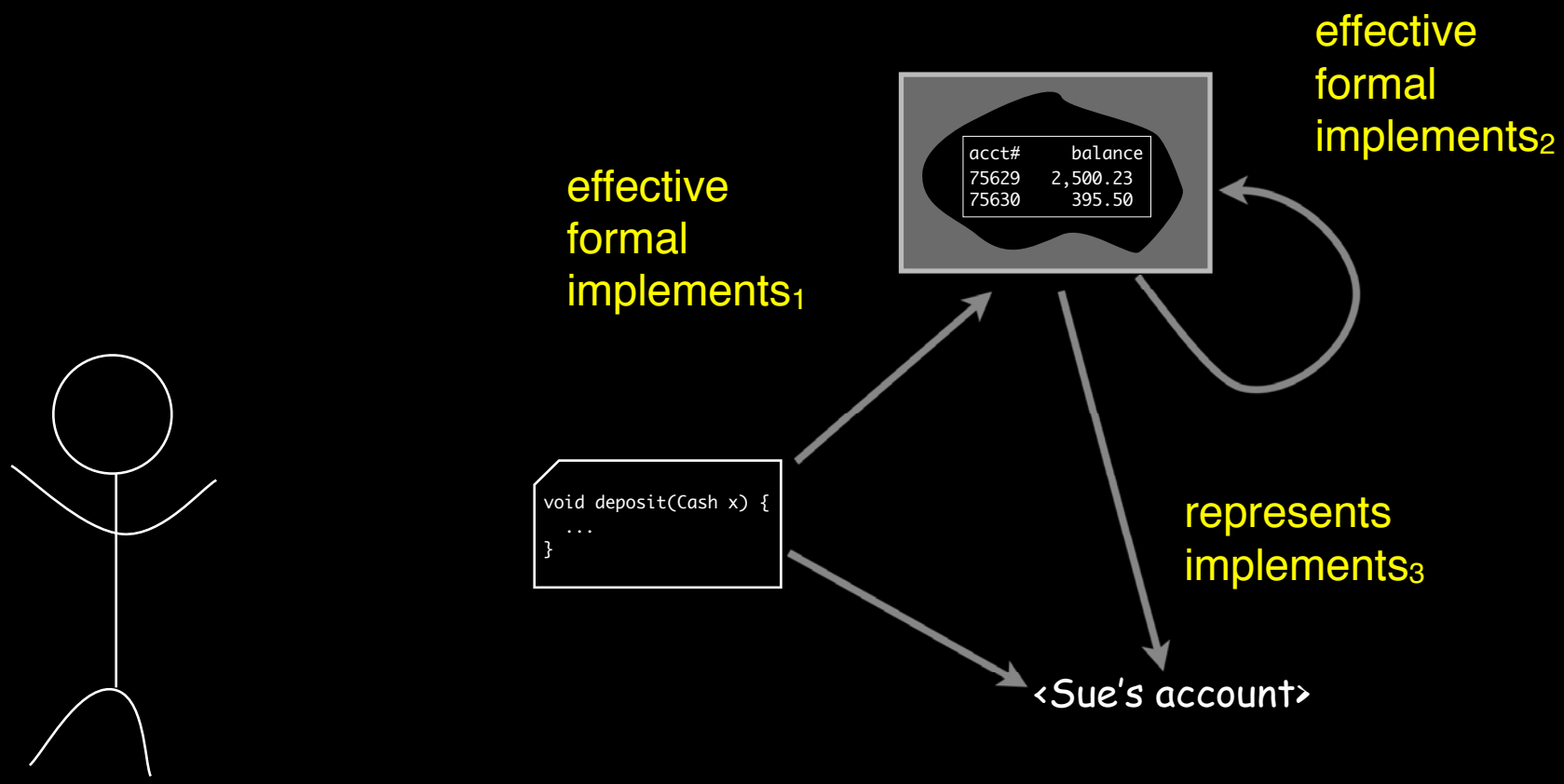
```
void deposit(Cash x) {
  ...
}
```

formal

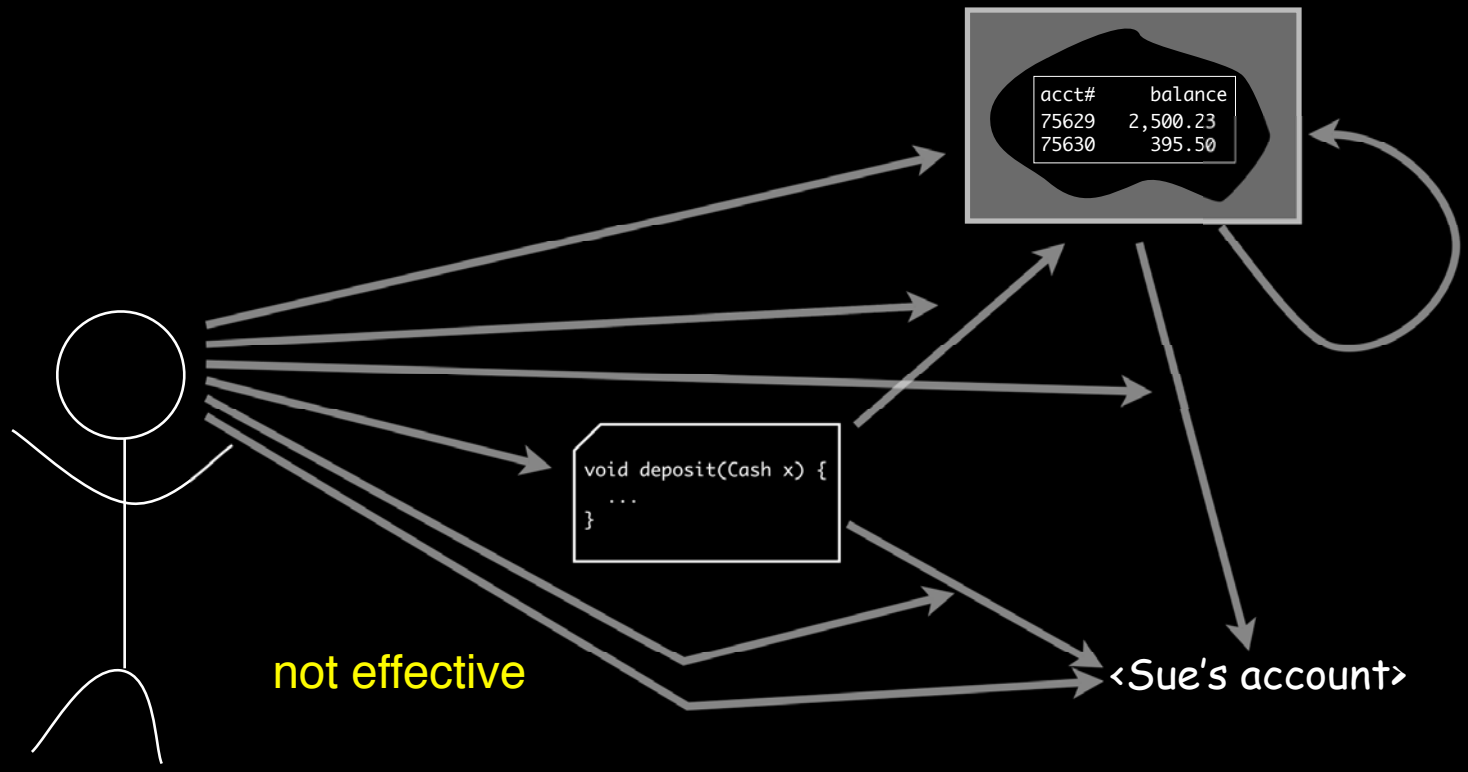
<Sue's account>

abstract/concrete,
formal/not...

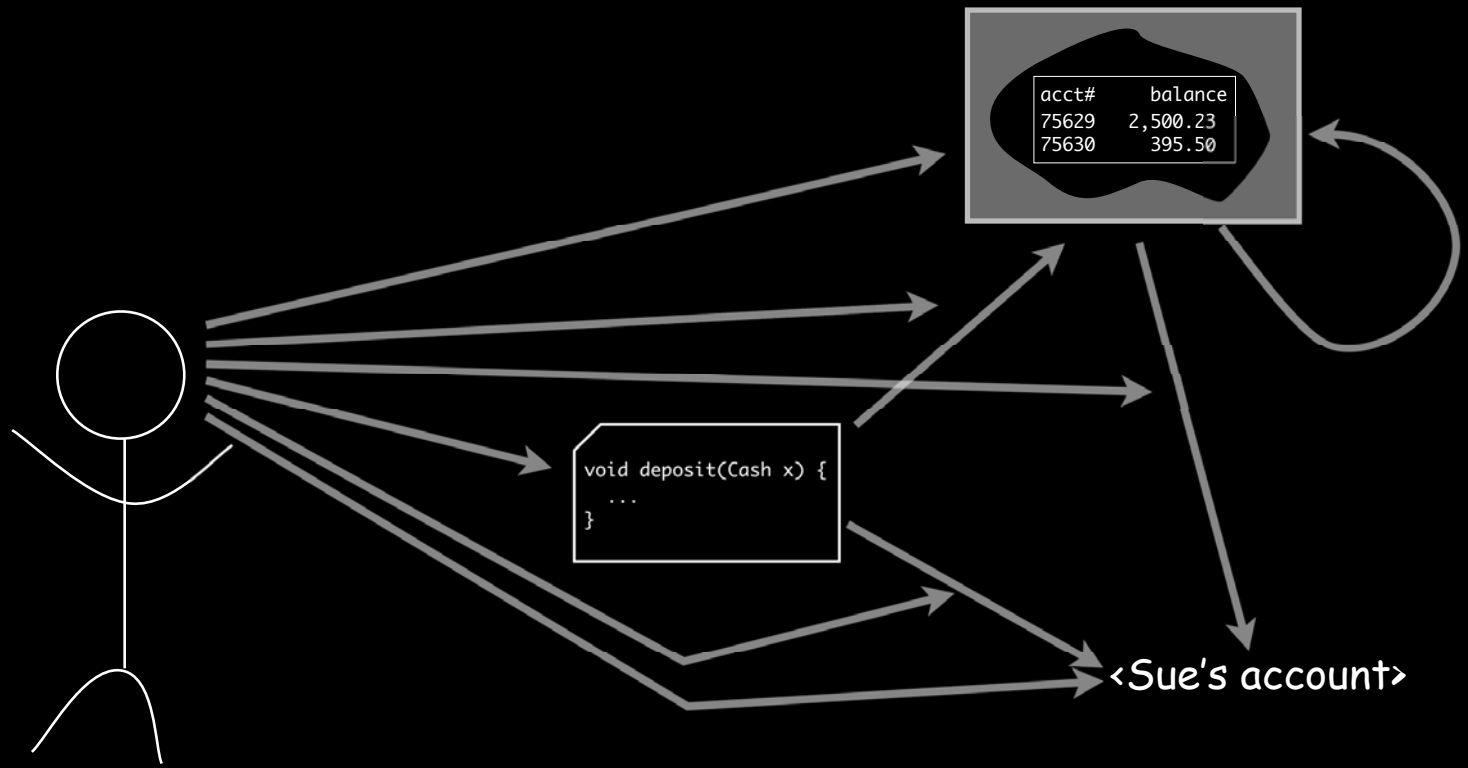
Developing A Trivial Banking System



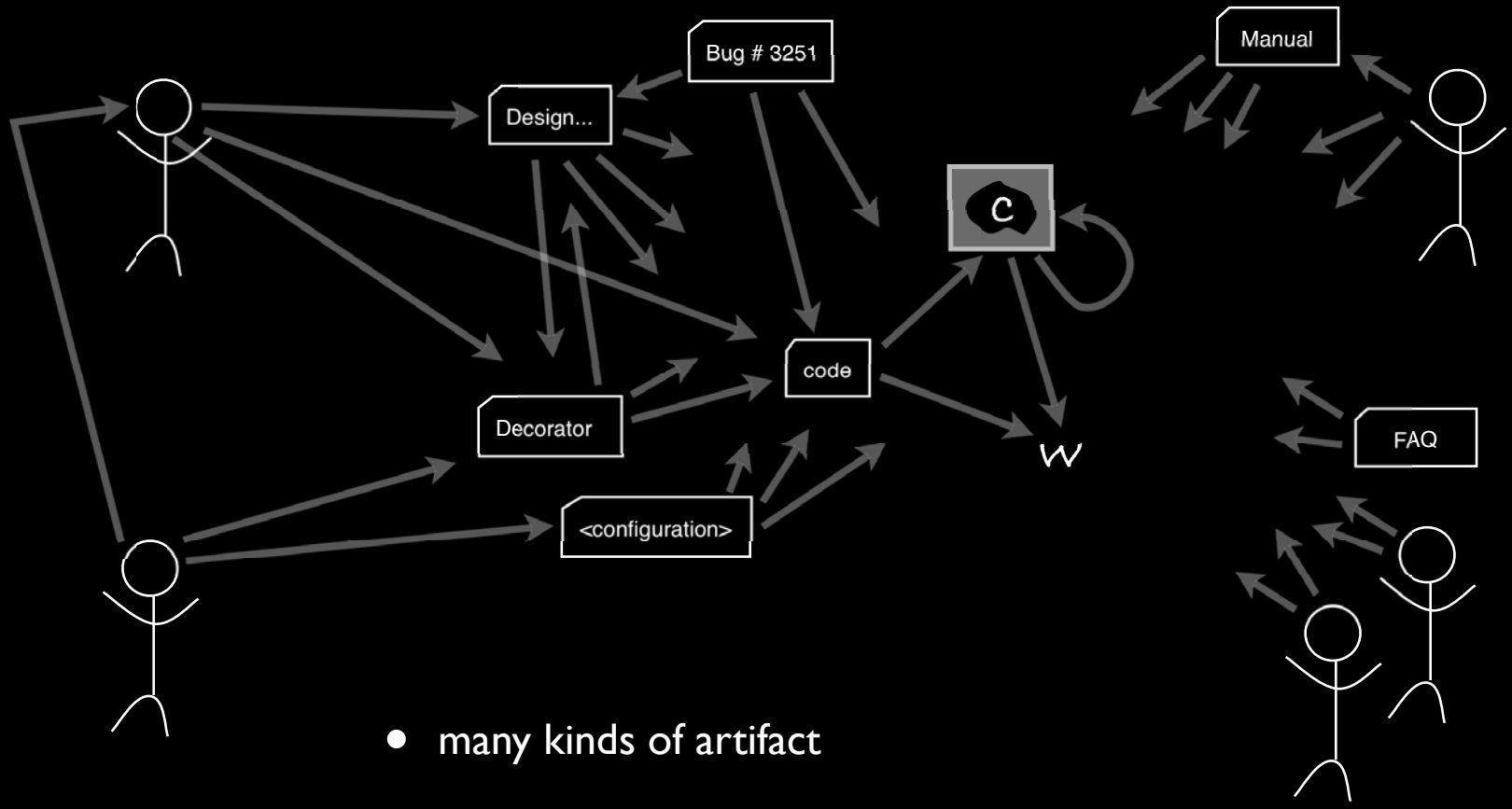
Developing A Trivial Banking System



Developing A Trivial Banking System

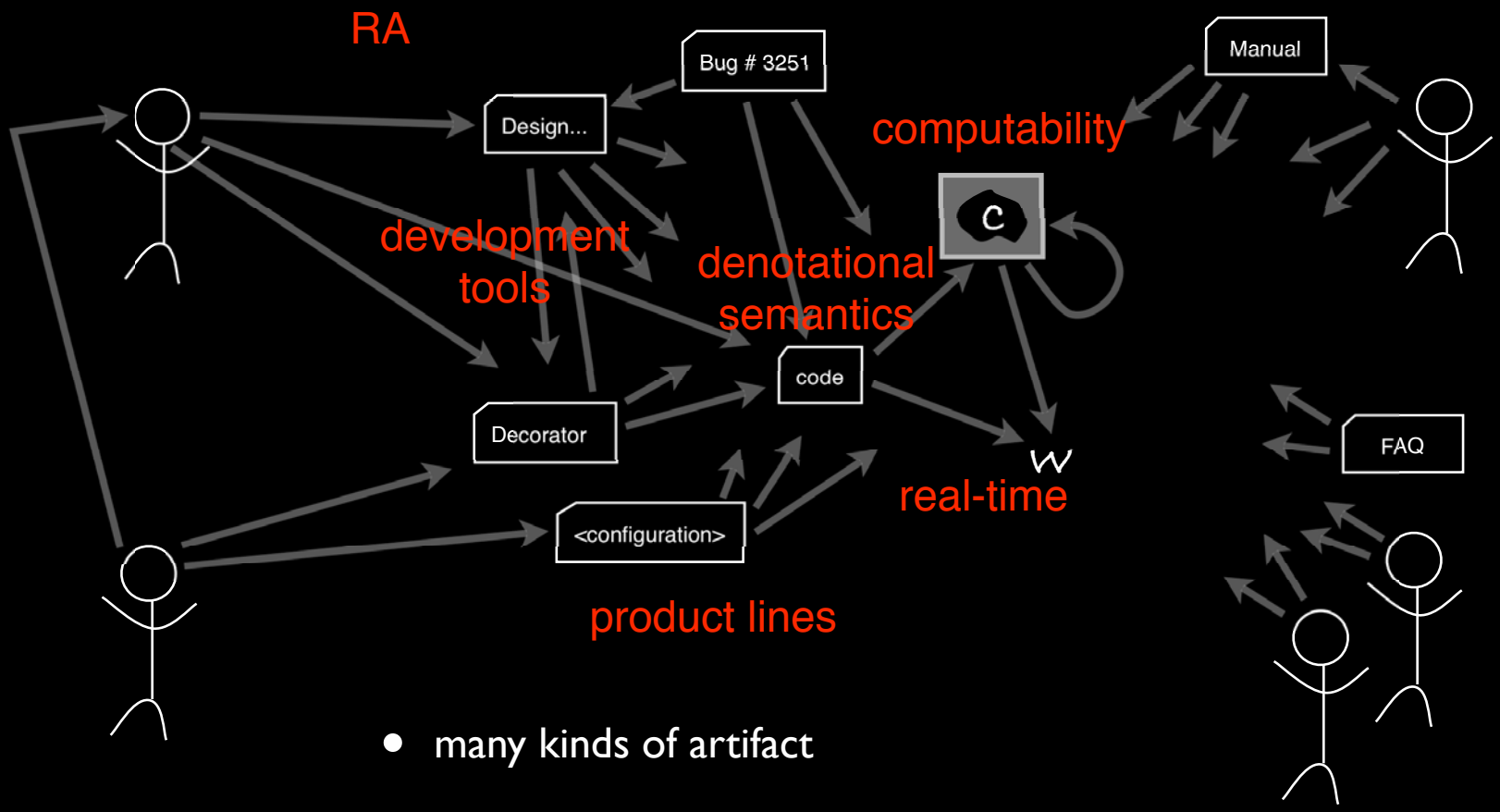


Developing Software



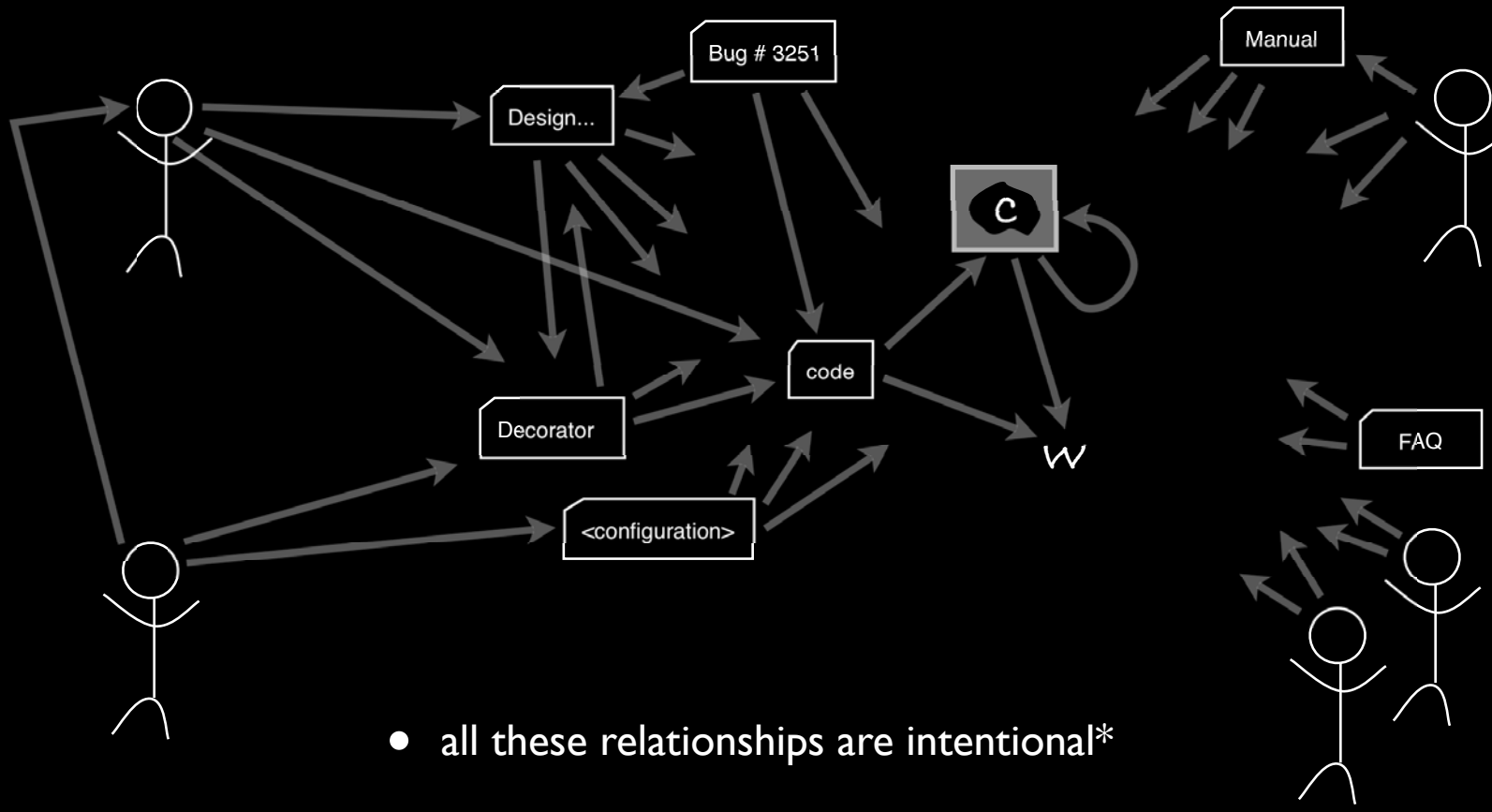
- many kinds of artifact
- diverse properties and relationships
- numerous theories and areas of work

Developing Software



- many kinds of artifact
- diverse properties and relationships
- numerous theories and areas of work

Intentionality As Spanning*



- all these relationships are intentional*
- crosses boundaries
- bridges divisions and highlights distinctions

The world is fundamentally characterized by an underlying flex or slop--a kind of slack or "play" that allows some bits to move about or adjust without much influencing, and without being much influenced by, other bits.
[Smith, *On the Origin of Objects*: 199]

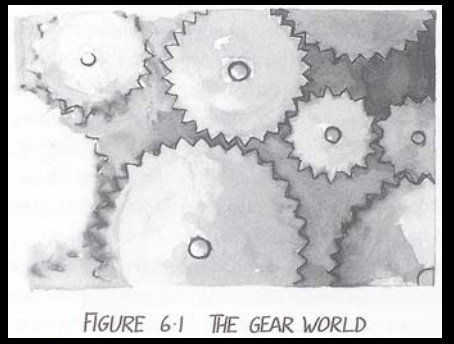


FIGURE 6·1 THE GEAR WORLD

Overall, my aim in this book is to show that the world's primordial flex or play does two crucial things: (i) establishes the problem that intentionality solves; and (ii) provides the wherewithal for its solution.
[Ibid.: 200]

That semantic reach exceeds effective grasp is essentially a theorem of this metaphysical account.
[Ibid.: 211]

...the world is not presumptively discrete--indeed it is as completely opposite of formal as it is possible to imagine. It is instead permeated by:

*1.*Indefiniteness [zest][1] *at the edges of given objects*, such as the boundaries of the region on the wall where I ask you to write your name...

*2.*Indefiniteness [zest] *between and among objects of the same type*, such as whether you are standing on this sand dune or the neighboring one

*3.*Indefiniteness [zest] *among different types*, such as among chutzpah, bravado, ego, self-confidence and brashness;

*4.*Indefiniteness [zest] *among the notions 'concept,' 'type,' and 'property'...*

*5.*Indefiniteness [zest] *between objects and the types they exemplify,* implying that the "instance-of" relation is itself approximate, contested, and potentially unstable--as for example in whether the headache you have this morning is the same one you had last night, or a different one of the same type; and similarly for patches of color, fog and "the rain"; and

*6.*Indefiniteness [zest] *between and among different realms of human endeavor,* such as the political, the social, the technical, the religious, the esthetic, the psychological, etc.

[Ibid: 324]

---

1. Page 324 has 'indefiniteness'; page 325 explains that 'zest' is a better word.

Registration

By ‘register’ I mean something like *parse, make sense of, find there to be, structure, take as being a certain way* -- even *carve the world into,* to use a familiar if outmoded phrase.

[Ibid.: 191]

...summarize three essential properties of [registration]:

1. Registration is the net activity that leads to (what we theorists register as) a conception of, or take on, or intentional attitude towards, the world as given or available--anyway as *world*.

2. Registration is originally neutral as to the appropriate locus, if any, of two essential subject/world splits: (i) that between registrar (subject) and what is registered (object), and (ii) that between subject and supporting community (people, instruments, practices, documents, culture, etc.).

3. Registration does not single out objects as a premier ontological category or class--or even, necessarily, require that objects count as a distinct ontological species.

[Ibid: 197]

Combined Themes

- "The world is as opposite of formal as it is possible to imagine"
  - no single right structure, or even ontology
  - abstractions are transient, shifting, negotiated
  - things are not formal at the bottom
- " they know what they are talking about in that way"
  - actions, including plan production are situated
  - objectivity is achieved rather than given, all language is indexical
  - things are not formal at the top

Radical Thesis

work with, have effective access to

- To get at higher-level (interesting?) issues
  - formality is not the foundational idea, and
  - layers of effective formality is not the right mechanism
- Effectiveness has to be more sloppy
  - negotiated, periodic, partial, evolving
  - built on something like registration
- Want a corresponding supporting theory

Some Objections

- This kind of stuff is hooey!
- This kind of stuff isn't hooey, but this version of it is
- Maybe this is true for people
  - but programs are engineered, they can and must be crisp
- This is the unavoidable difference between PL and tools
- The thing you are talking about is already happening
- Interesting, but can’t be made to do work

Registration in Emacs

Note that EMACS, a popular text and programming editor, derives much of its power from supporting multiple simultaneous “takes” on the string of characters in its buffer, in just the way suggested here. One command can view the buffer as a Lisp program definition; another, as a linear sequence of characters; another, as bracketed or parenthesized region. In order to support these multiple simultaneous views, EMACS in effect “lets go” of its parse of the buffer after every single keystroke, and re-parses all over the next time a key is struck--possibly with respect to an wholly different grammar.
[Smith, Origin of Objects: 48]

Registration in Emacs

```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```

Registration in Emacs

```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```

ctrl-k (kill line) command registers text at point as first line and rest

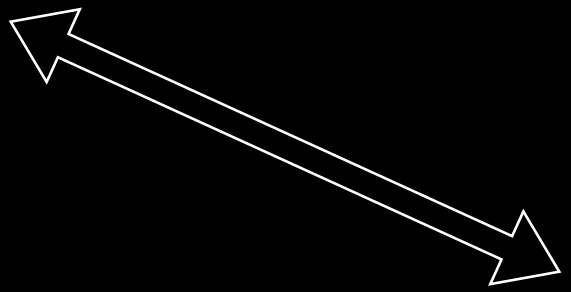
```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```

Registration in Emacs

```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```

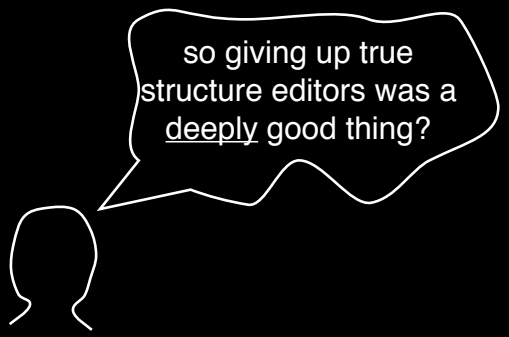
ctrl-k (kill line) command registers text at point as first line and rest

```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```



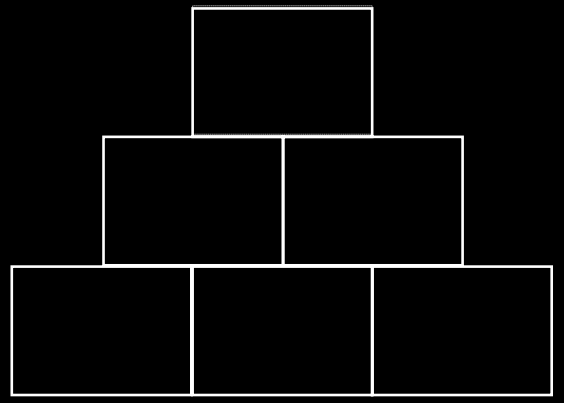
ctrl-meta-k (kill sexp) registers text at point as first sexp and rest

```
;; make a new point
(define make-point
  (lambda (x y)
    (list 'point x y)))

;; getters
(define point-x second)
(define point-y third)
```



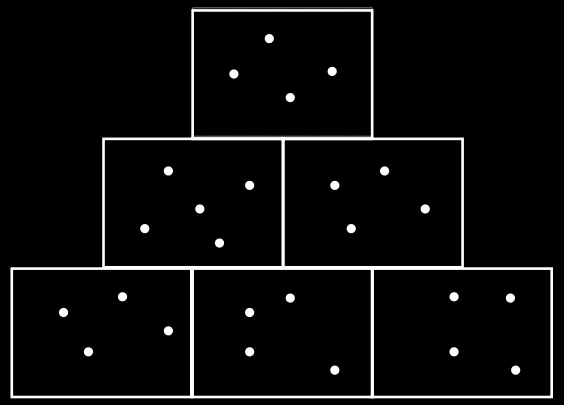
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



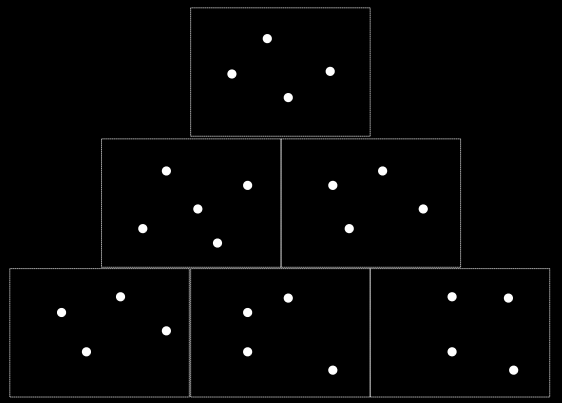
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



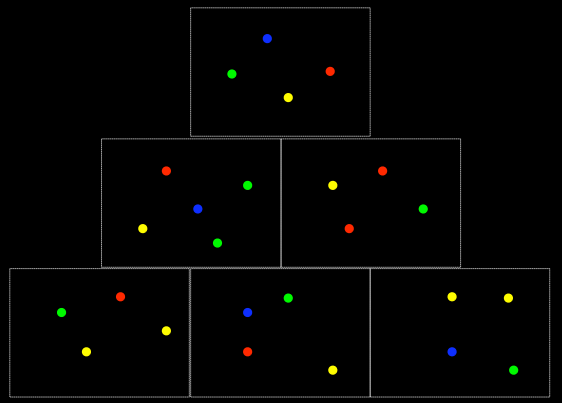
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



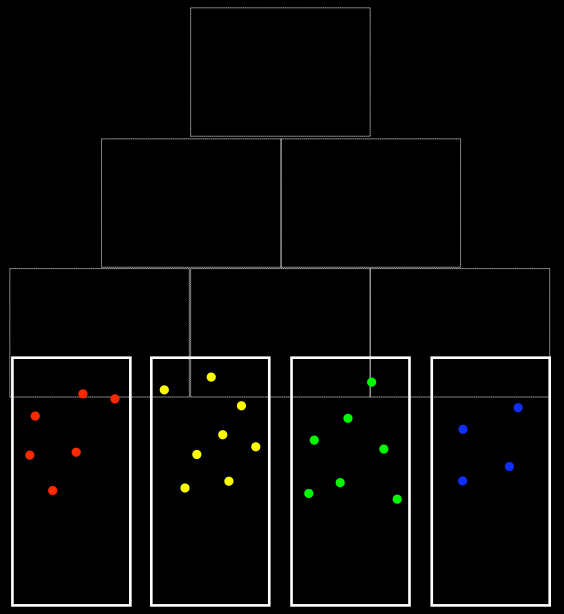
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



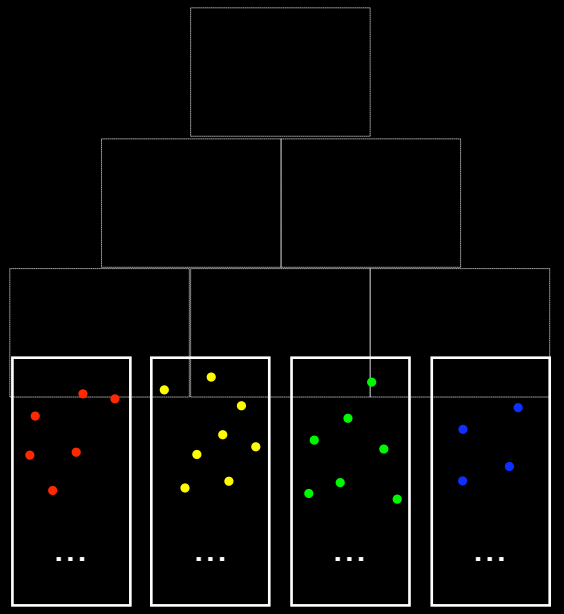
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



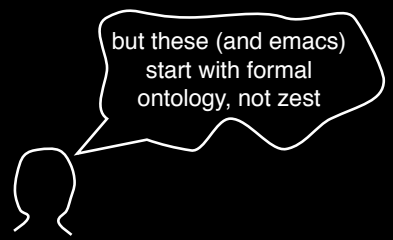
Join Point Mechanisms

- (Start with classic structure)
- Atomize to see sub-elements
- Dissolve structure boundaries
- Register new structural elements
  - using sub-elements properties
  - and context dependence
- Provide effective access via new structure



JPMs In Familiar AOP Systems

- AspectJ pointcuts and advice
  - points in execution flow, pointcuts, advice
- AspectJ intertype declarations
  - class member declarations, signatures, declarations
- Hyper/J
  - class members (broadly), signature patterns, slice, compose



Registration-Based Effectiveness

- Not formal?
  - but digitality at the bottom implies formalilty
- Focus on zest and other properties
  - tradeoff between crispness and effectiveness
  - negotiated abstractions, effectiveness
  - transient registrations
  - external semantics
- Also see Rok Sosic's dissertation

Design Pattern Rational Graphs

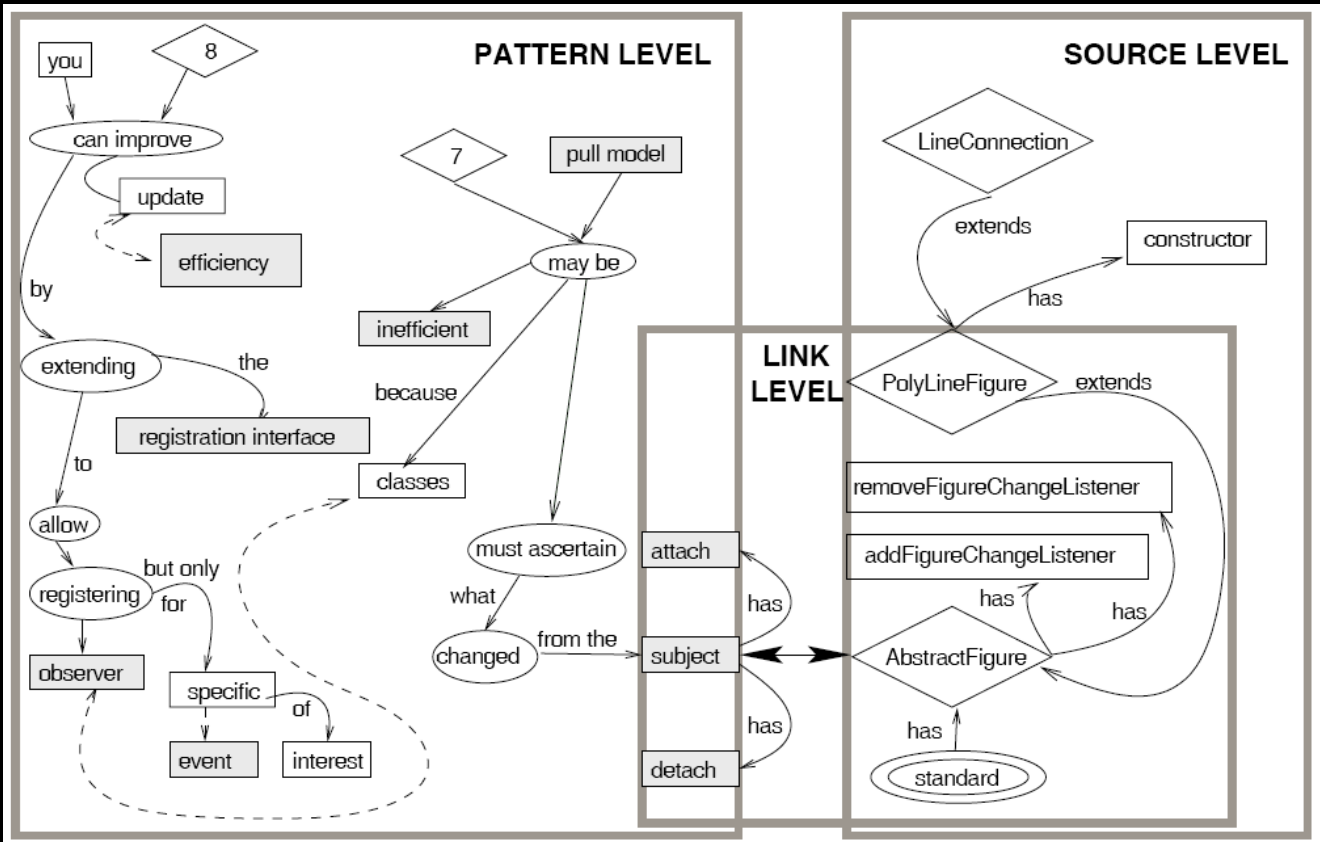


Figure 2. Three queries on a DPRG of OBSERVER/JHotDraw: a regular expression query for **efficien, a link level expansion of *subject*, and a source level expansion of `PolyLineFigure`**

[Baniassad, Murphy, Schwanninger. ICSE '03]

Role Based Refactoring

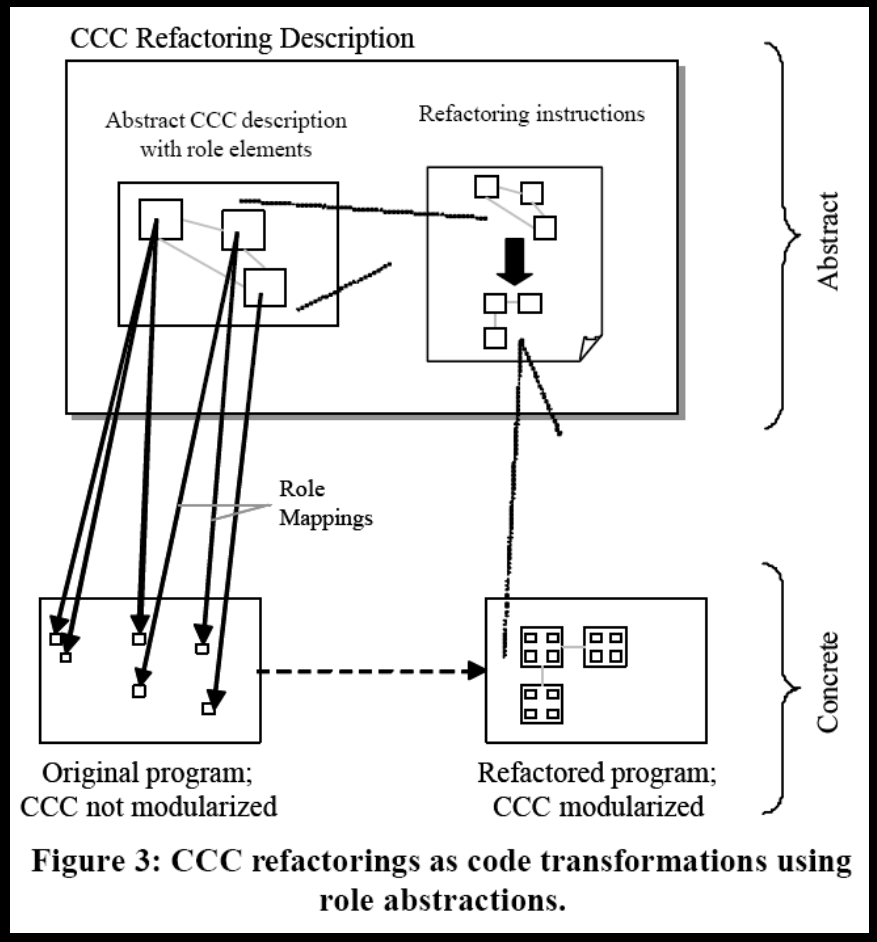
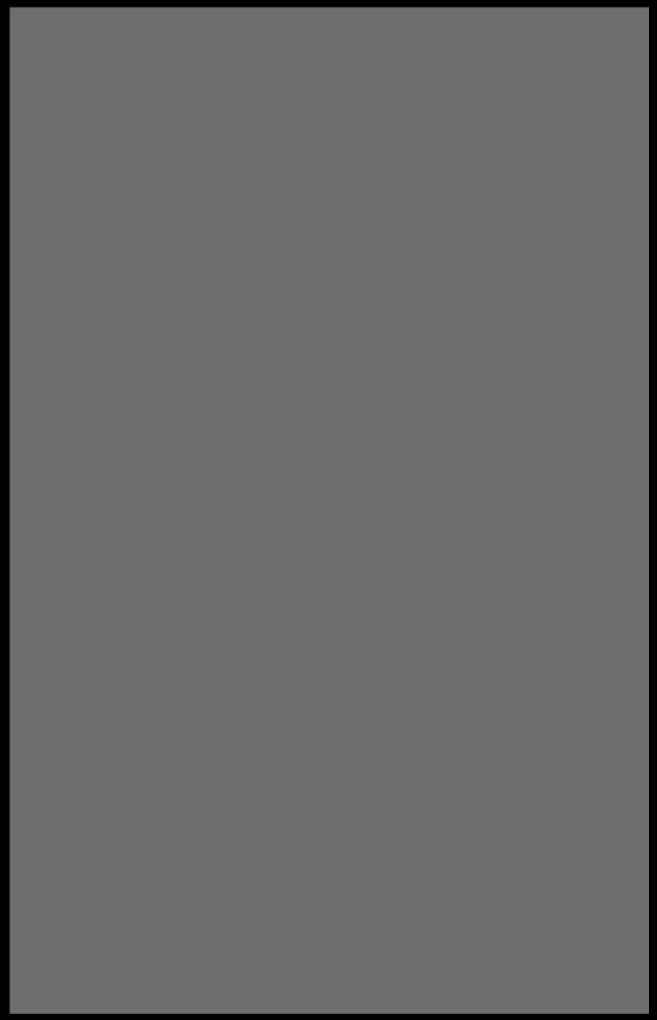


Figure 3: CCC refactorings as code transformations using role abstractions.

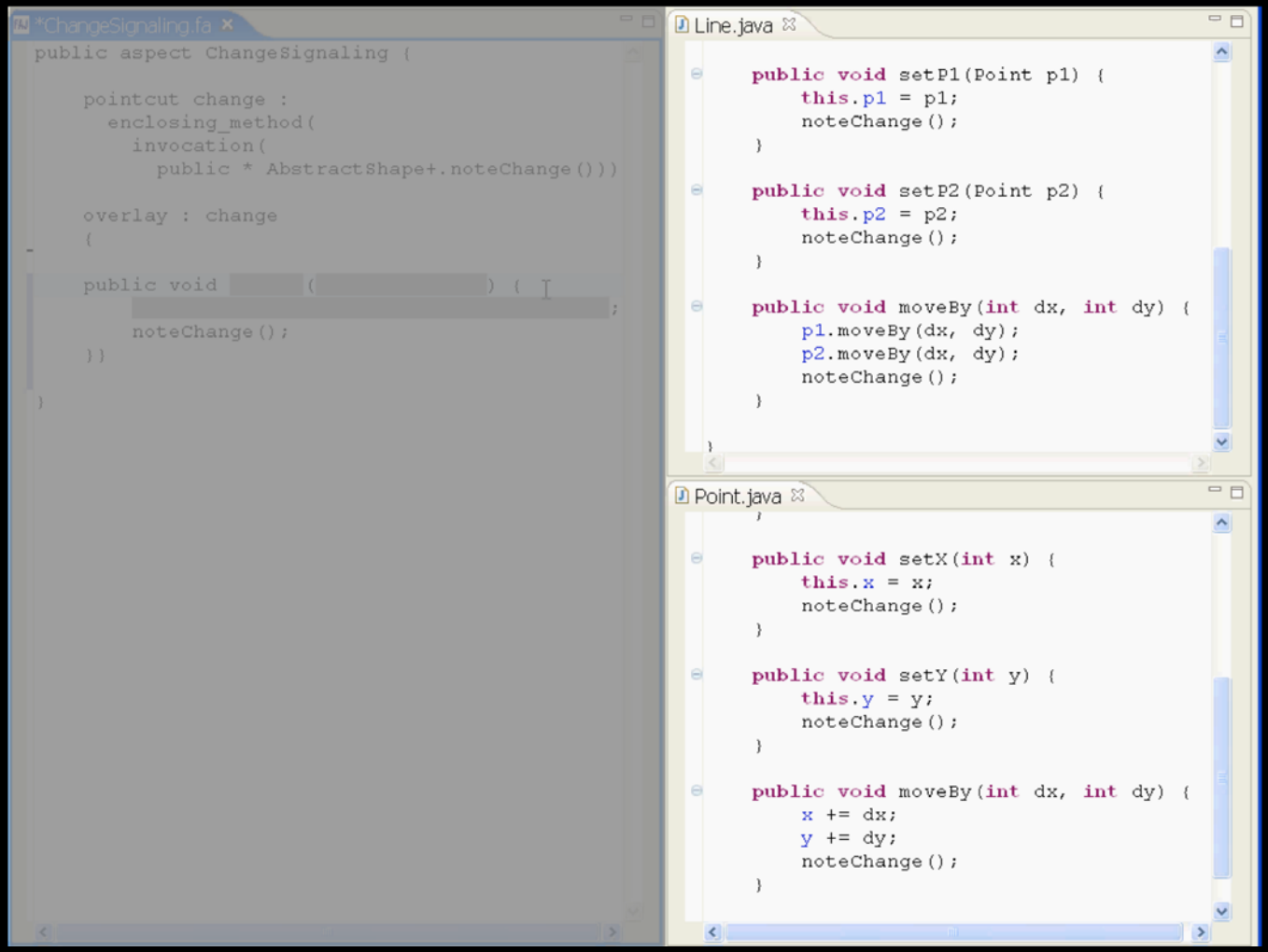
[Hannemann, Murphy, Kiczales. AOSD '05]

Fluid AOP Prototype



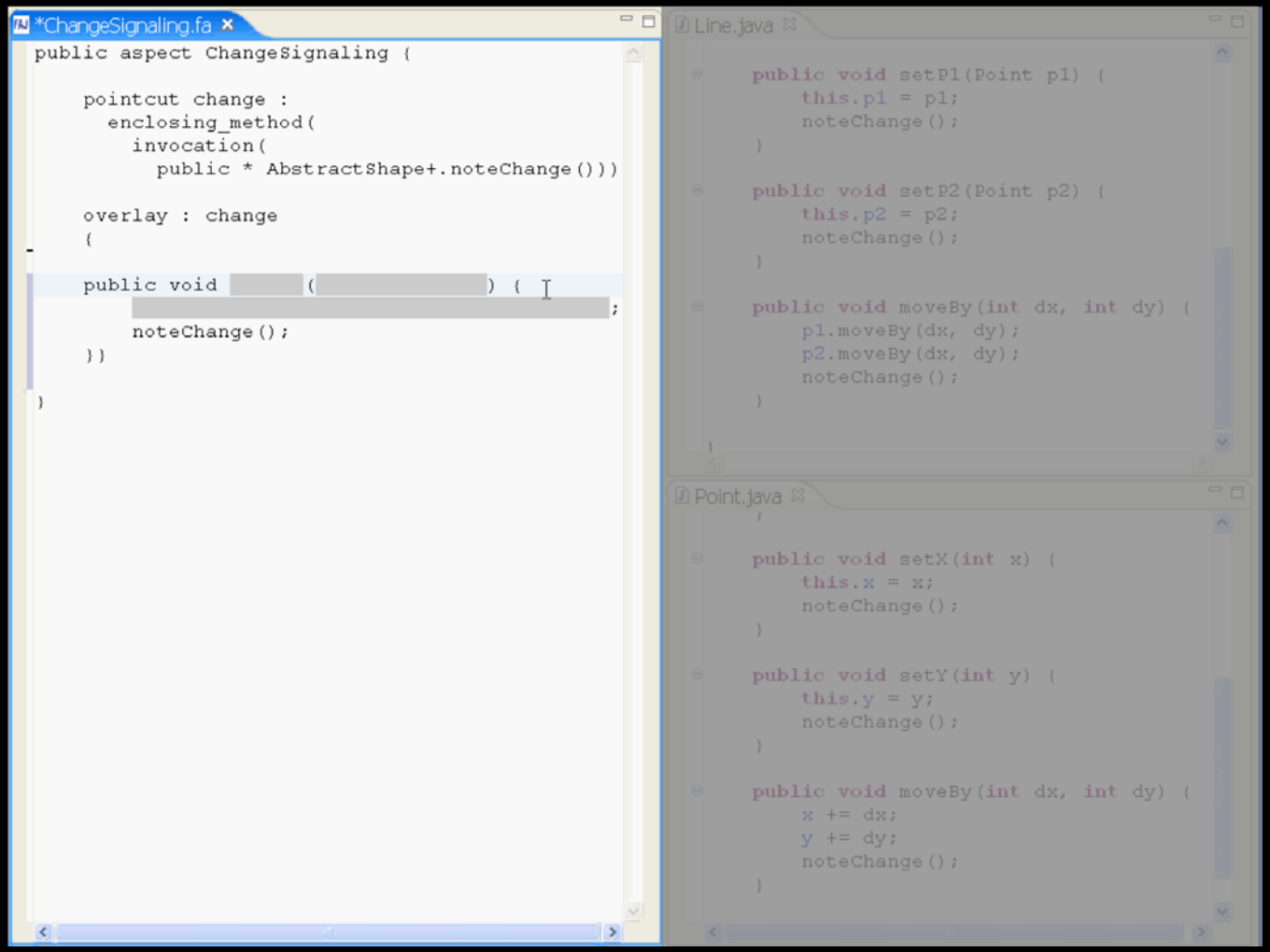
[Hon, Kiczales. OOPLSA ’06 Demonstration]

Fluid AOP Prototype



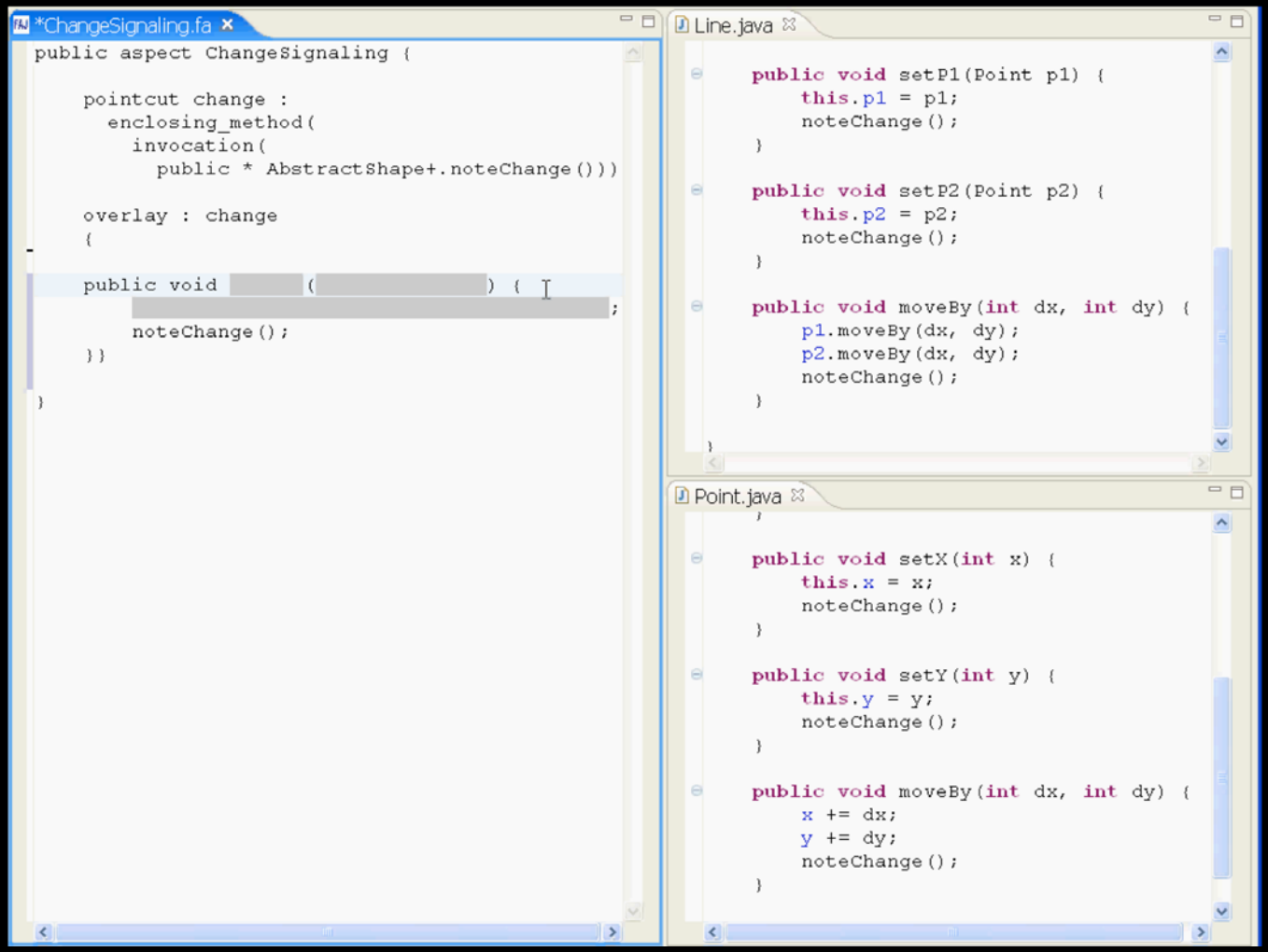
[Hon, Kiczales. OOPLSA '06 Demonstration]

Fluid AOP Prototype



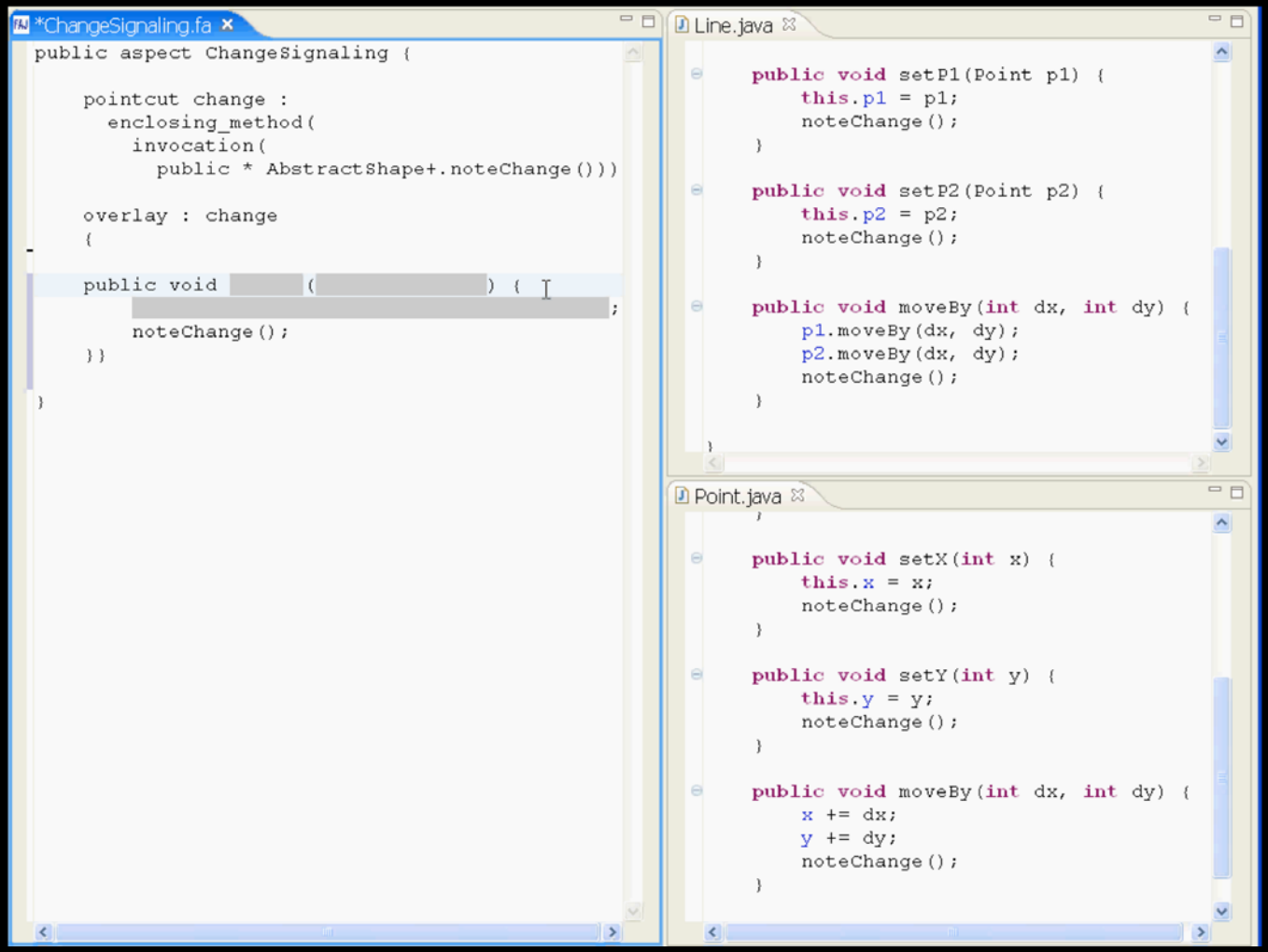
[Hon, Kiczales. OOPLSA '06 Demonstration]

Fluid AOP Prototype



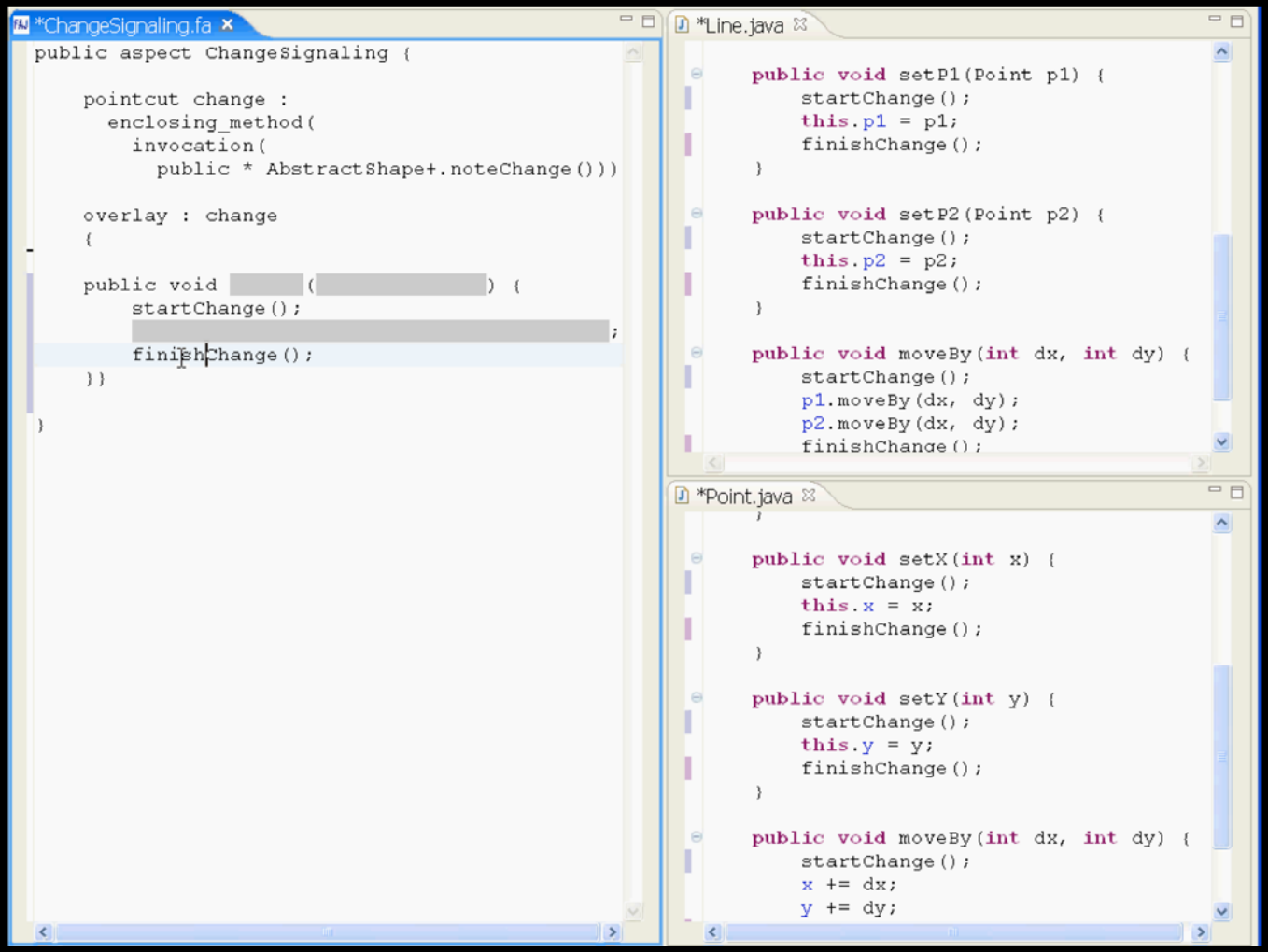
[Hon, Kiczales. OOPLSA '06 Demonstration]

Fluid AOP Prototype



[Hon, Kiczales. OOPLSA '06 Demonstration]

Fluid AOP Prototype



[Hon, Kiczales. OOPLSA '06 Demonstration]

Example Summary

- External socially-mediated semantics
- Effectiveness via external semantics
- Negotiated, transient abstraction
- Abstraction/effectiveness tradeoff

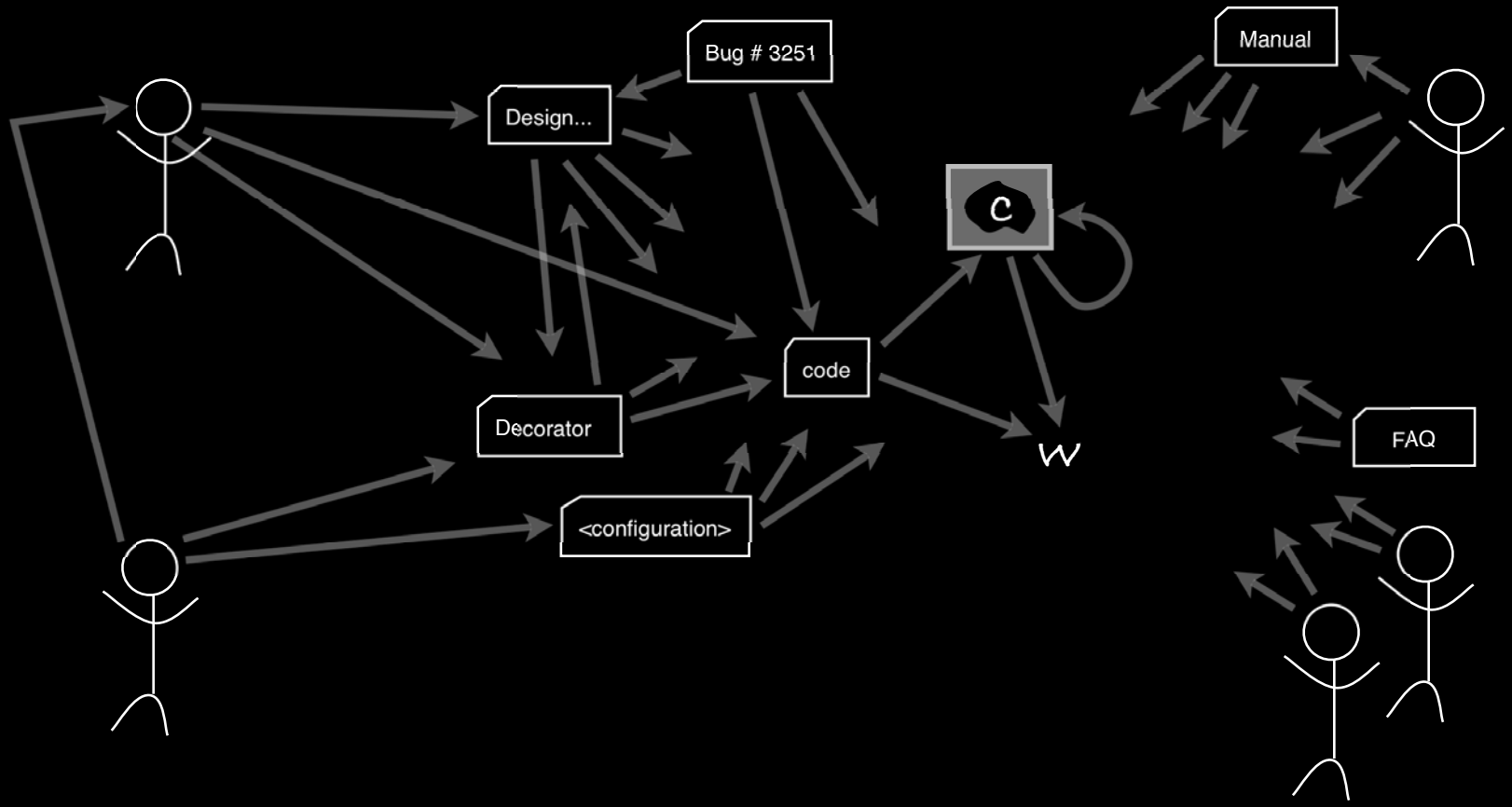
The Challenge

As much as twenty years ago, members of the knowledge representation community in artificial intelligence, and also a number of researchers working on databases, began to wrestle with many of the same problems as now face object-oriented system designers. In part as a reaction to the insuperable difficulties they encountered, many people in the knowledge representation community abandoned the idea that a system's ontological categories (those in terms of which it deals with its primary subject matter) should be explicitly represented at all. Instead they viewed them as emerging in constant and dynamic renegotiation with the with the environments in which these systems play or are deployed. It is interesting to speculate on how the mainstream programming community will rise to this challenge of developing external, social and negotiated categories.
[Smith, Origin of Objects: 48]

Recap

- Patterns and Mylyn
  - flexibility with high-level abstracions & situation particulars
- Programming languages
  - effective, but problems with flexibility
- Conflicting foundational stories
  - our current foundations
  - intentionality, ethnomethodology

Intentionality As Spanning*



Summary

- Software development is rich with intentional relationships
- Intentional things are not formal all the way down
  - at the 'highest levels' -- ethnomethodology, indexicality of language
  - at the 'lowest levels' -- O^3, 'flex and slop', registration
- Effective higher-level abstractions must cope with this limit of formality
- Suggested what 'registration-based' effectiveness might be
  - characteristics (negotiated, social, external...)
  - old and new examples (patterns, Mylyn, fluid AOP, DPRG, RBR)
- A new 'foundations of software development' can and should be built this way