How do you naturally express your programming ideas?
Embracing Informality and Ambiguity

*Kenneth C. Arnold* and *Henry Lieberman*

*Software Agents* · *Commonsense Computing Initiative*  
*MIT Media Lab* · *Mind Machine Project*
the world is complex, so

Say what you know, and not more.

By forcing us to use precise vocabulary, software development makes us say too much.
Proposal In Brief

- Development tools should let us talk about our programs in many ways.
- Different degrees of formality
- Different degrees of ambiguity
- Interactively develop rich mappings: an ecosystem of different representations

“"You don't understand anything until you learn it more than one way.”"  
Marvin Minsky
When we can be informal and ambiguous, software will:

- Be easier to develop
- Adapt to new situations
- Behave reasonably when things go wrong
Clarifying Terms...

- Specs and tests are formal but ambiguous.
- Formal = controlled semantics; incl. programming languages.
- Managed, intentional ambiguity
Developing Software is Easier

• Computer helps earlier in design
• Formality forces premature commitment
• Can communicate in more natural language (not just write code that looks like English)
• Let computer help evaluate multiple ways of solving a problem

When we can be informal,
How?

- Understand the informal representations
- Map to increasingly formal specs, code, and tests
- Learn by reading existing code (just like good programmers do today)
- Semi-automatic, example-driven iterative refinement
Informal, Ambiguous Natural Language
(goal, high-level characteristics)

Common Sense and Domain Specific Knowledge

Implementation Characteristics

Concrete Implementations
demo in Scratch opportunities for photo/video editing, PIM, etc.

Joint inference using Blending

ConceptNet

track
move with arrow keys
projectile motion
bouncing

Child doForever > forward_
Sibling forward_ ~ pointTowards_
Clump [forward_ pointTowards_]
Presence doForever
Containment FlagHat doForever

ProcedureSpace
When we can be informal,

**Software will adapt to new situations**

- Software in the complex and rich real world
- Real life is more nuanced than any programmer can plan (or just wait, it’ll change)
- User considerations impact even backend
- Software must act appropriately in a variety of situations
How?

• Software that knows about everyday life
• crowdsourcing (ConceptNet)
• hand-coding (Cyc)
• learn from sensors, social media, etc.
• Context-appropriate behavior: continuously evaluate against informal representations
When we can be informal,

Software will fail intelligently

• If software always failed in expected ways, explicit failure handling would be merely tedious

• Enumerating failure scenarios impossible in complex systems.

• A system that only works one way can fail in many ways
Failing Intelligently

• try alternative approaches
• central problem-solving knowledgebase
• informed defaults for unspecified details
• suggest possible failure scenarios and reasonable courses of action
Discuss...

- When we can be informal and ambiguous, software will
  - Be easier to develop
  - Adapt to new situations
  - Behave reasonably when things go wrong

- Development tools should permit artifacts at different degrees of formality and ambiguity, and interactively develop rich mappings among them