Dr. Jones: A Software Archaeologist's Magic Lens

Mark A. Foltz

November 15, 2001

The Problem

- Program comprehension is time consuming and cognitively taxing.
- Documentation is scarce and out-of-date.
- Source code browsing environments are inadequate.

Outline

- User Studies and Results
- Supporting Navigation and Capturing Abstractions
- Situating Dr. Jones

User Studies of Java Browsing

- Observe software browsing behavior.
- Collect browsing strategies and rationale.
- Collect frustrations with and wishes for browsing tools.

User Studies: Subjects

- 12 sessions, 12 subjects, 2 programs/session
- Computer science graduate students
- 3 months to 5 years of Java experience
- Longest program worked on: 6 classes to 100K+ LoC
- 480 minutes of browsing behavior

User Studies: Methods

- Subjects given an unfamiliar program organized as hypertext [Demo]
- Subject prompted while browsing to capture rationale
- Maintenance tasks
- End-of-session interviews

User Studies: Conditions

	Single Pane Structure	Two Pane Method+Callees	Two Pane Method+Callers
Solitaire 3 classes, 265 LoC	N=7	N=2	N=1
Client-Server 12 classes, 206 LoC, dead code	N=4	N=3	N=2

User Studies: Strategies

1. Name Guessing (10)
2. Starting from main() (9)
3. Critical-Method (5)
4. Sequential (4)
5. Top-Down (3)
6. Bottom-Up (2)
7. Tracing (2)

User Studies: Results

Wishes

1. Dependency information (3)
2. Keyword searching (2)
3. Dead/abstract code elimination (2)
4. Diagramming (2)

Frustrations

1. Unhelpful variable names (6)
2. Lack of comments (6)

User Studies: Take-Away

- Consistent browsing strategies
  - In terms of program structure and behavior
  - Can assist user navigation in the program
- Names are crucial
  - Guide programmer's choices across strategies
  - Would renaming be useful?

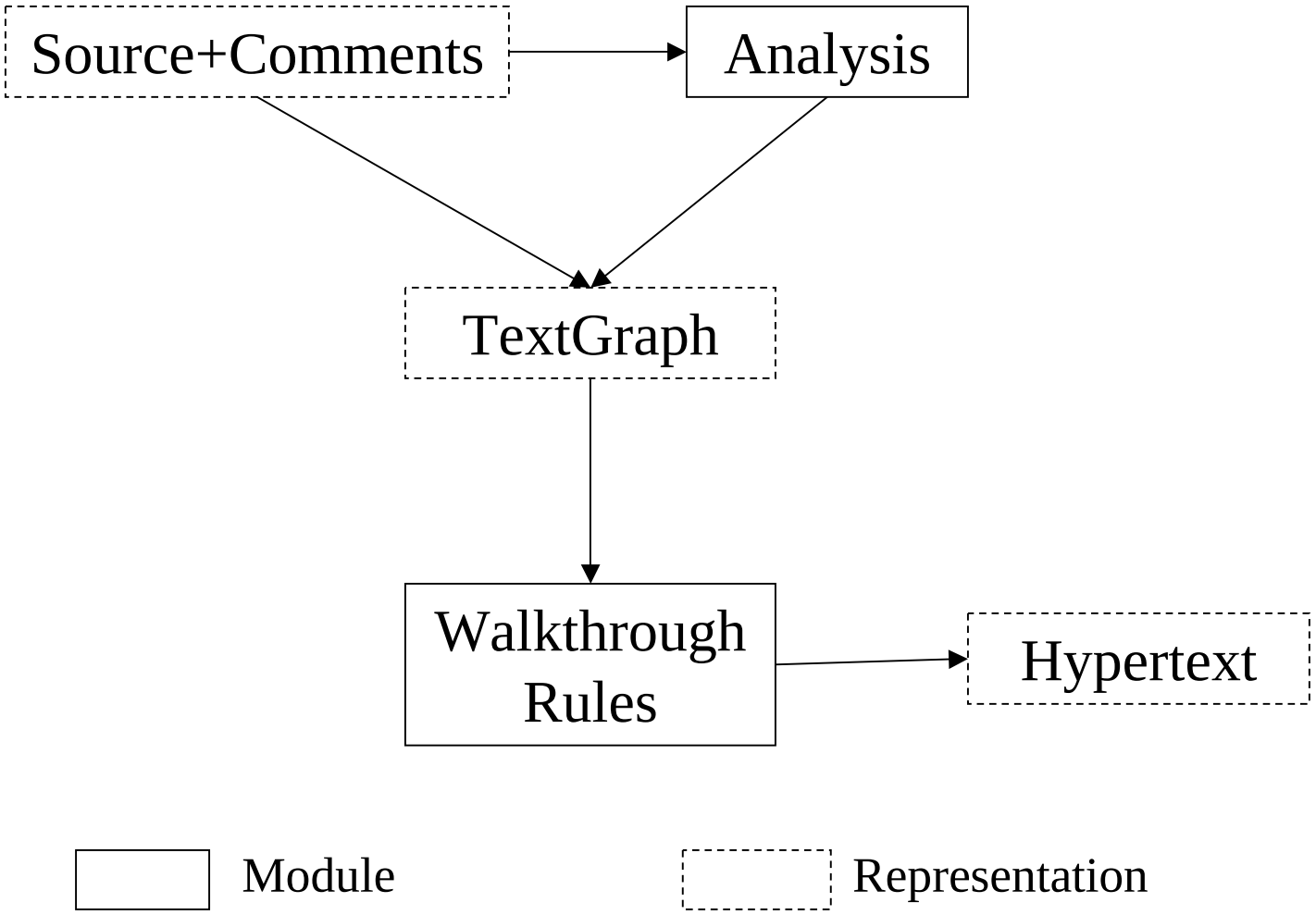
Outline

- ✓ User Studies and Results
- ➢ Supporting Navigation and Capturing Abstractions
- ➢ Situating Dr. Jones

Dr. Jones: Supporting Navigation

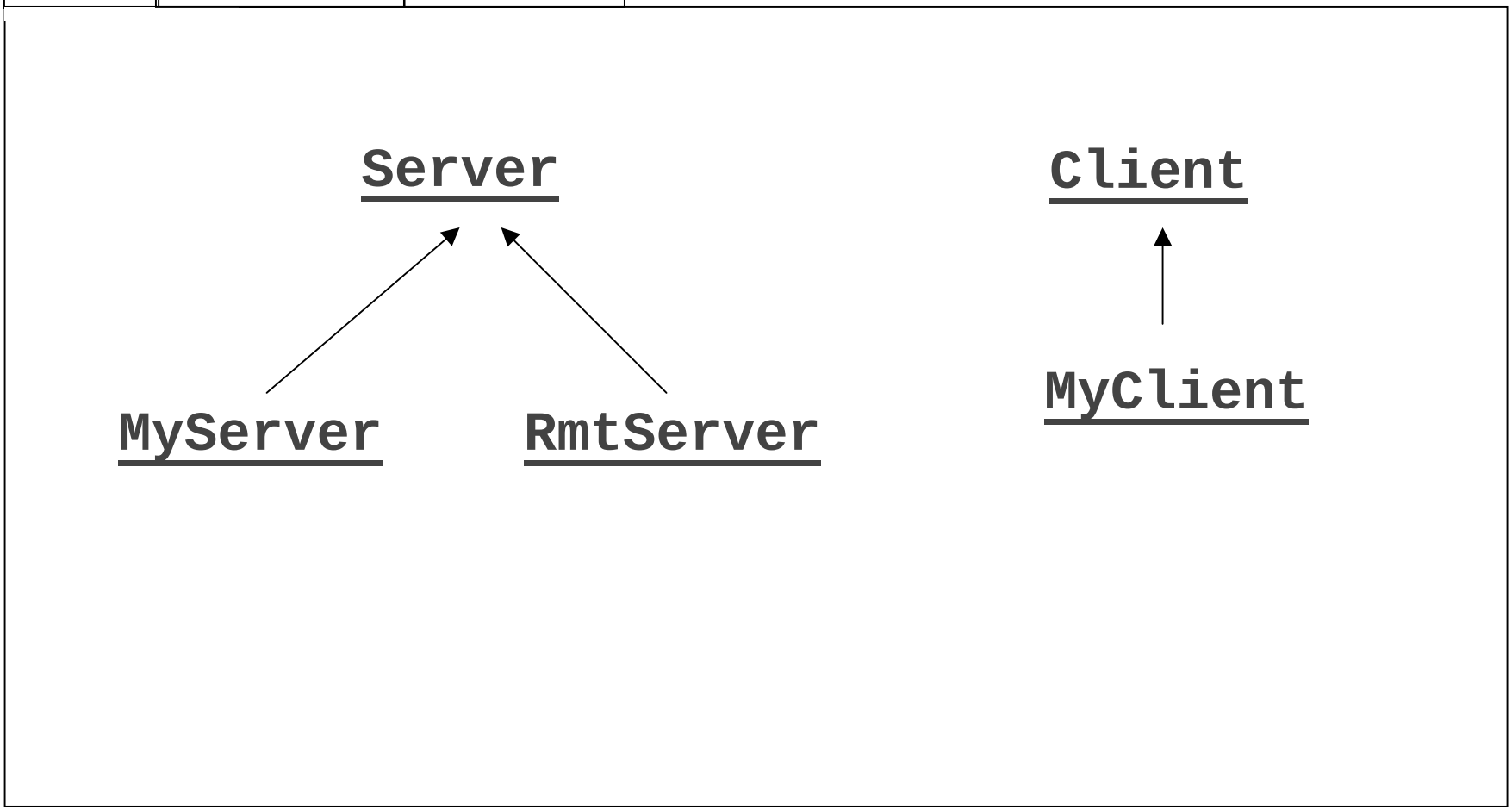
- ✓ Goal: Rapidly prototype software browsers
- ✓ Build a graph of the program text (a *TextGraph*)
  - – Nodes contain source and comments.
- ➢ Add edges to the TextGraph from program dependencies
- ➢ Support navigation by creating a walkthrough

Dr. Jones: Supporting Navigation



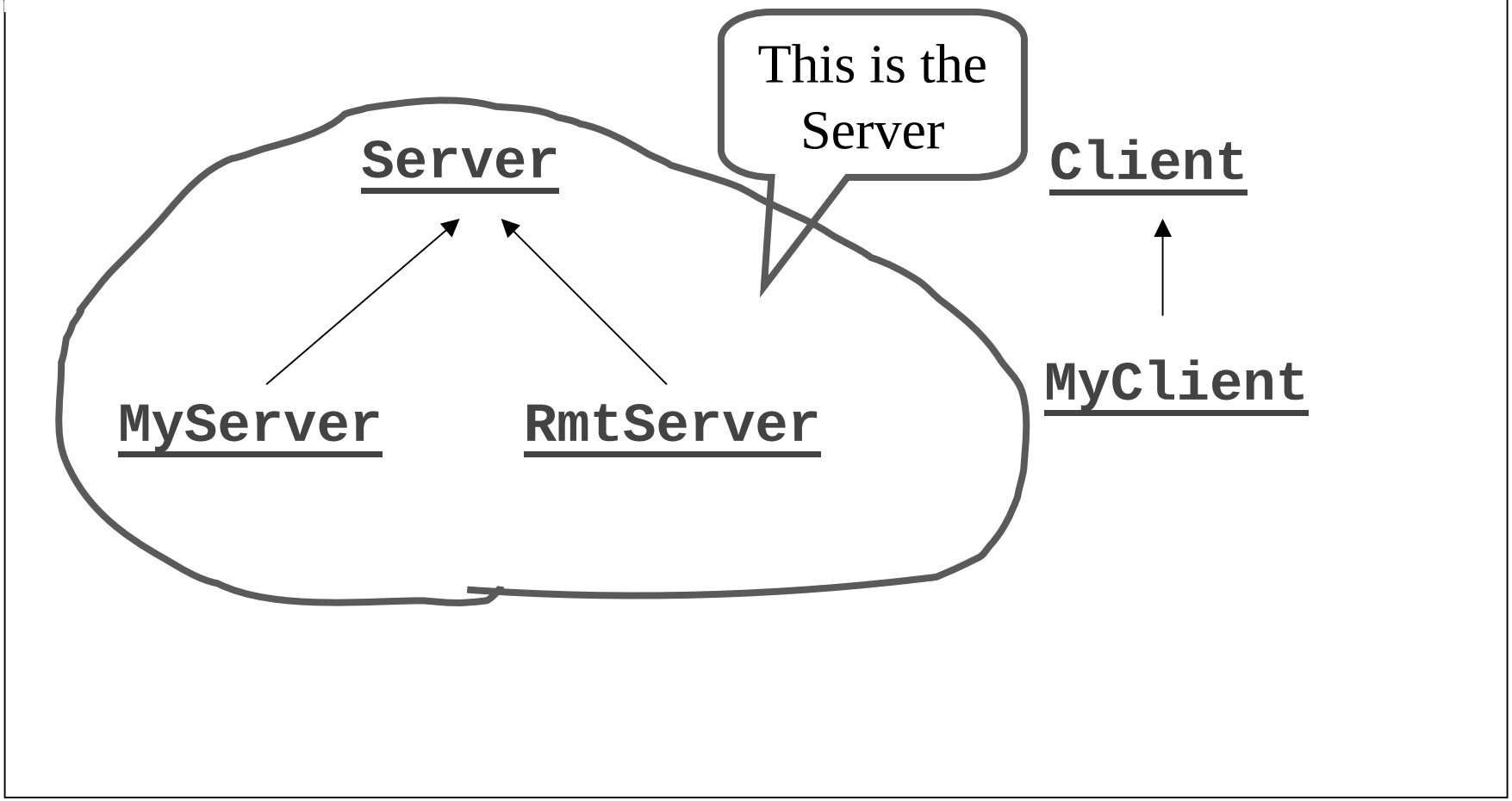
Dr. Jones: Capturing Abstractions

Class | Sequence | Object



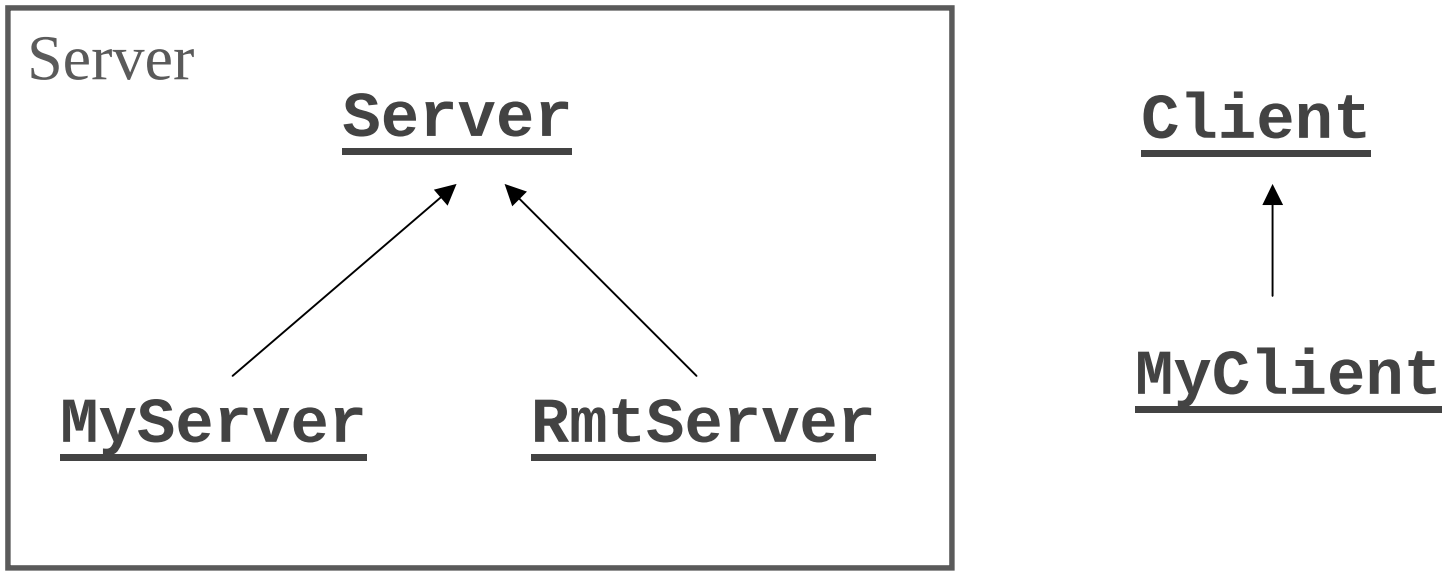
Dr. Jones: Capturing Abstractions

Class	Sequence	Object

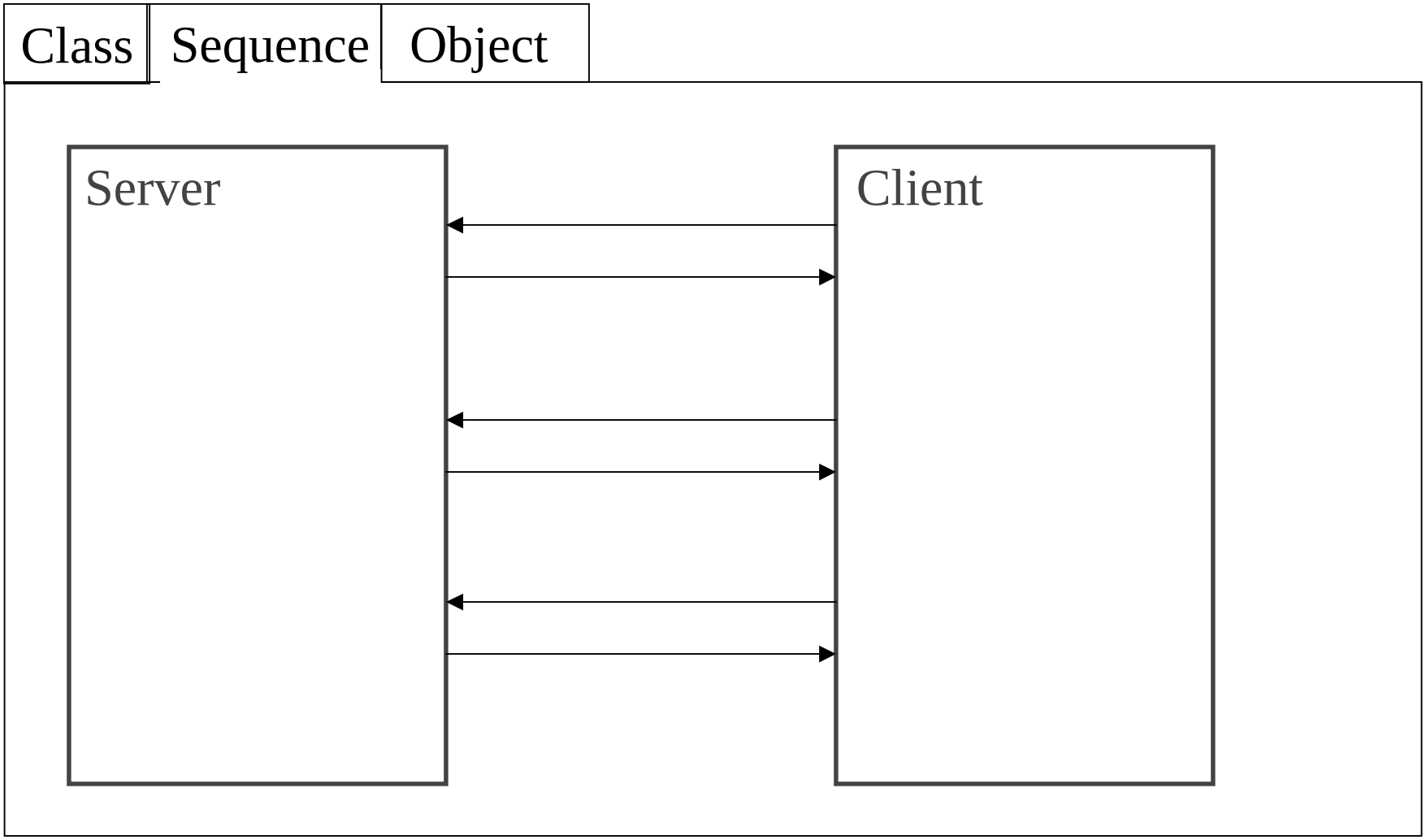


Dr. Jones: Capturing Abstractions

Class | Sequence | Object



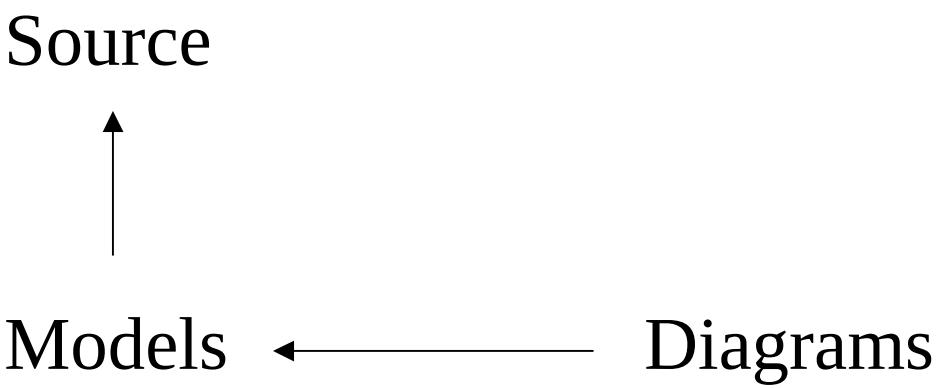
Dr. Jones: Capturing Abstractions



Outline

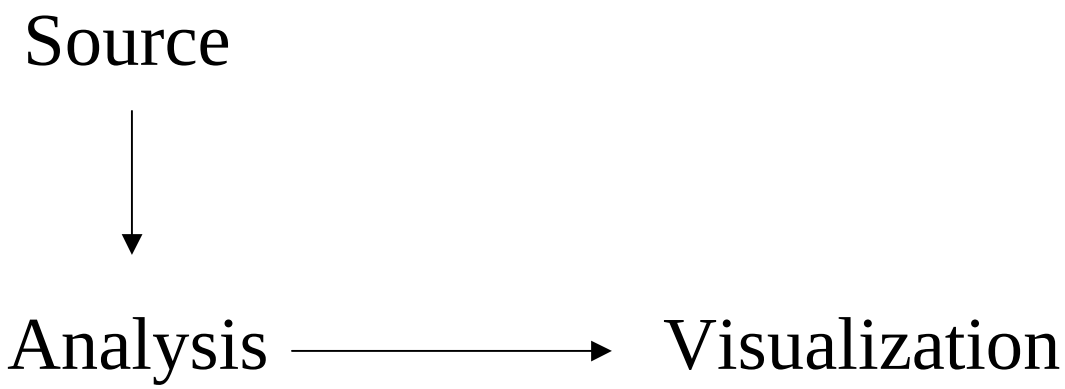
- ✓ User Studies and Results
- ✓ Supporting Navigation and Capturing Abstractions
- ➢ Situating Dr. Jones

Software Design



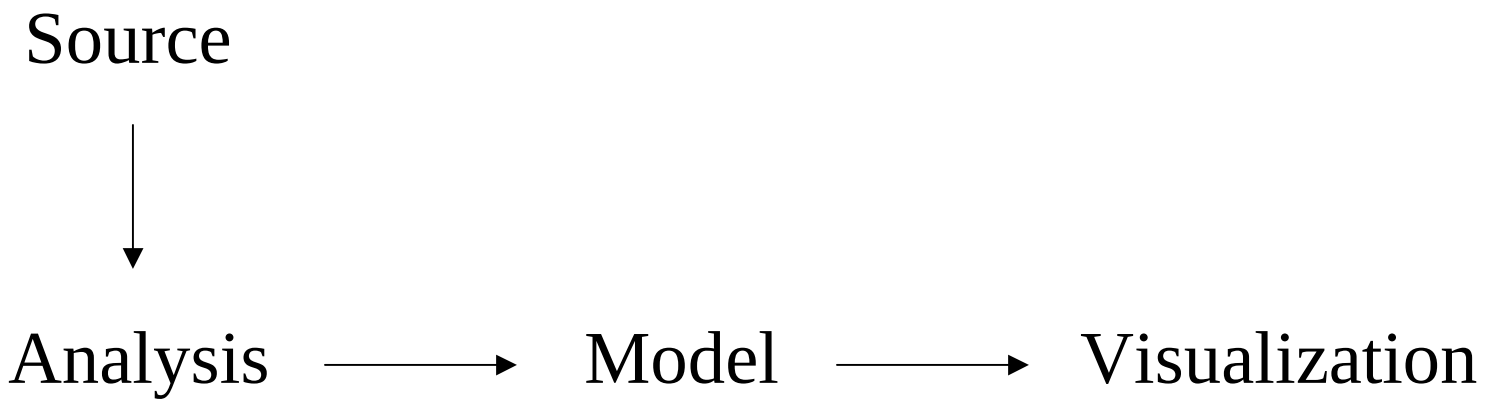
Booch Diagrams (Booch 1968), UML (1997-2001),
Z (Spivey 92), Alloy (Jackson 00)
Design Patterns (Gamma et al. 1995)
Rational Rose, TogetherSoft

Reverse Engineering



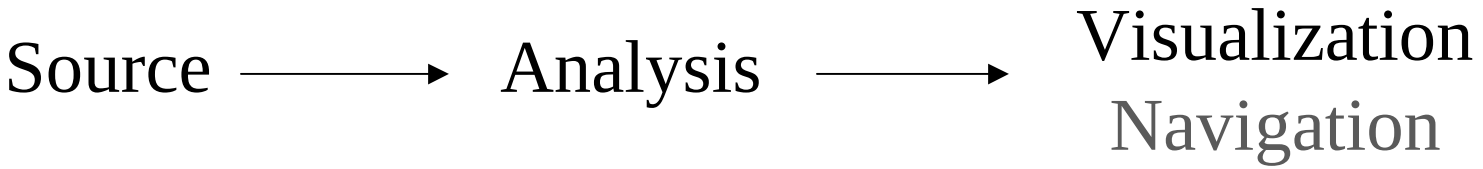
Pictoral Slicing (Jackson and Rollins 1994)
SeeSoft (Eick 1994), Aspect Browser (Griswold et al. 2001)
RiGi/SHRiMP (Müller, Storey, et al. 1992, 95, 96, 97)
CodeSurfer (GrammaTech)

Model-Based Reverse Engineering



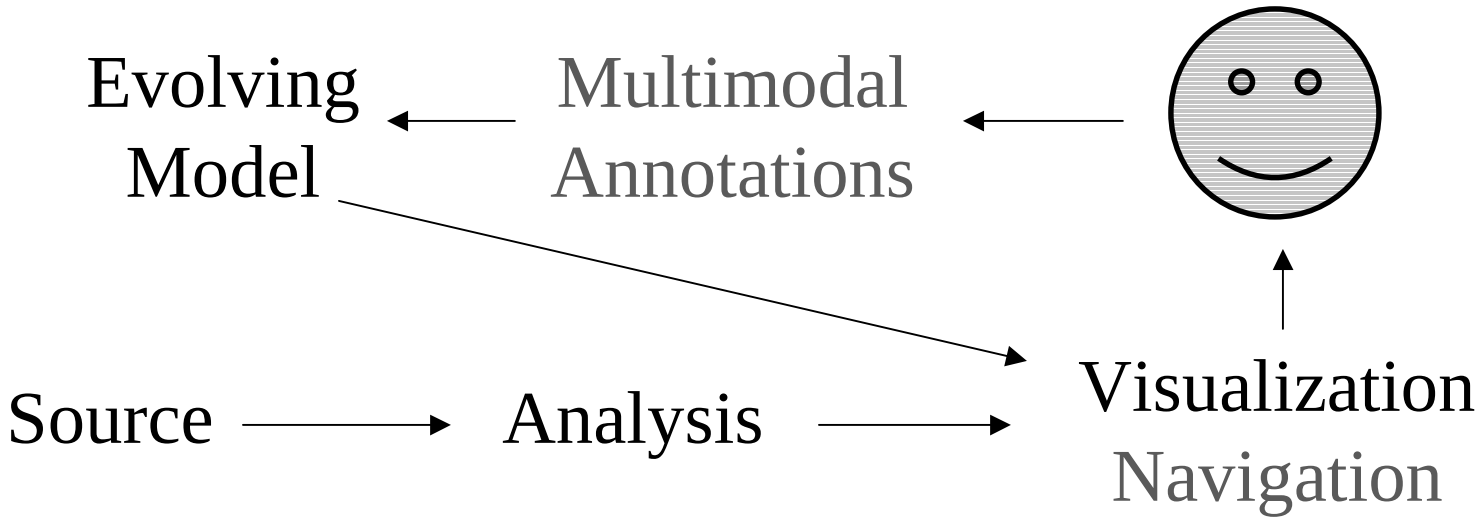
Programmer's Apprentice (Wills 1992)
Visualizing Design Patterns (Lange and Nakamura 1995)
Pattern Visualization (Schauer and Keller 1998)

Dr. Jones: Closing the Loop



Contribution #1:
A model of source code navigation that is explicit and programmable.

Dr. Jones: Closing the Loop



Contribution #2:
Supporting the natural capture of abstractions as they are discovered.

Software Reflexion Models
(Murphy, Notkin, and Sullivan 1995, 2001)

How to look at software?

Literate Prog. Knuth	Javadoc Dr. Jones	Seesoft
Linear One size fits all	Constrained Incremental	Random Access Overviews Lack of detail

Dr. Jones: Contributions and Challenges

- A set of heuristics that create effective walkthroughs.
- Multimodal annotation of software diagrams.
- *TextGraph*: Program representation grounded in the raw source, integrating programmer and automatic annotations.

Points for Discussion

- Visualizing software evolution
- Round-trip engineering
- Extreme and/or Pair Programming
- Other languages
- Invariant extraction (Daikon)
- Grungers vs. Architects
- What are useful views of a program?