

# Talking to Objects in Natural Language

## Toward Semantic Tools for Exploratory Programming

Christoph Thiede, Marcel Taeumel,  
Lukas Böhme, Robert Hirschfeld

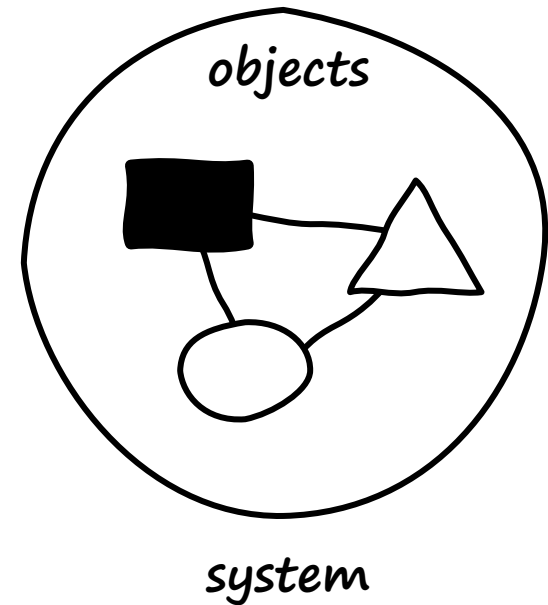
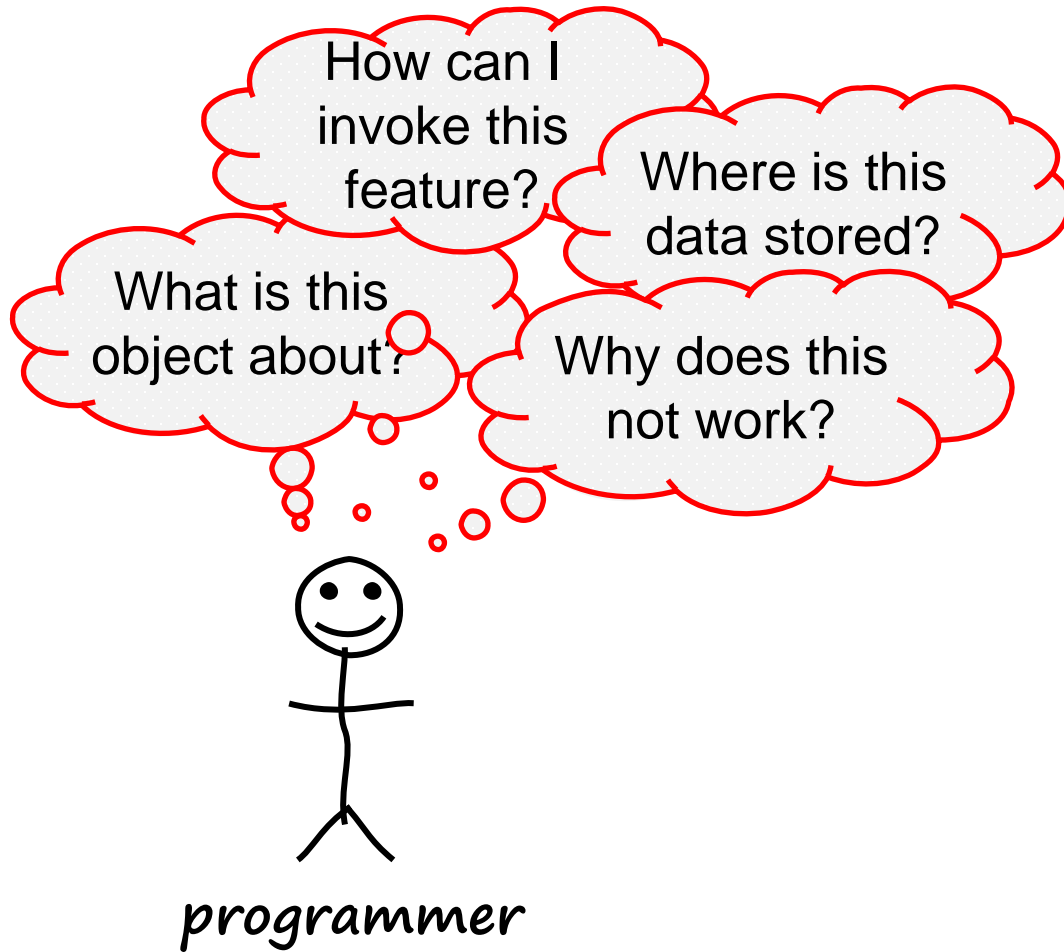
Software Architecture Group

Hasso Plattner Institute, Potsdam, Germany

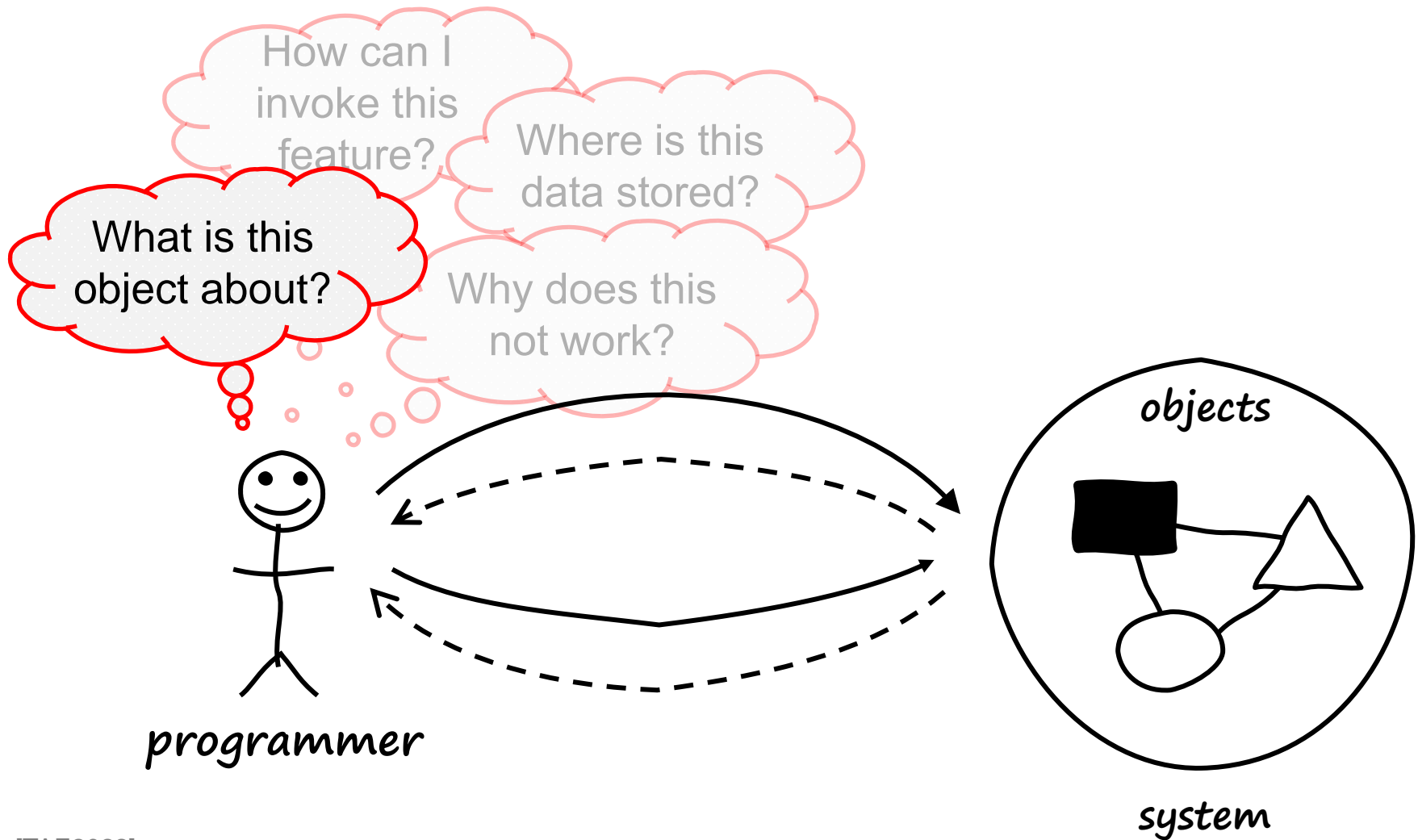
[hpi.uni-potsdam.de/swa](https://hpi.uni-potsdam.de/swa)

# Motivation

# Motivation

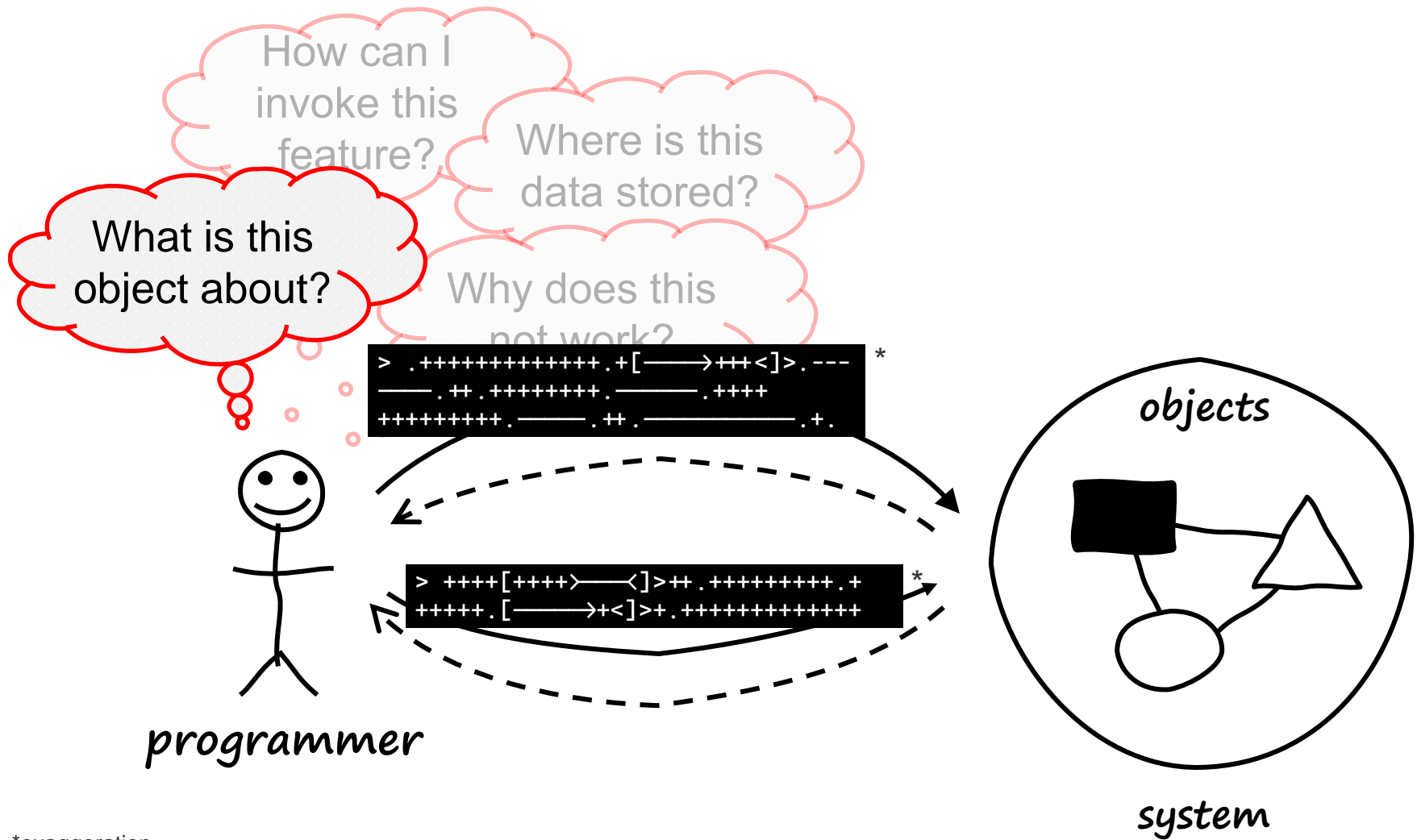


# Motivation



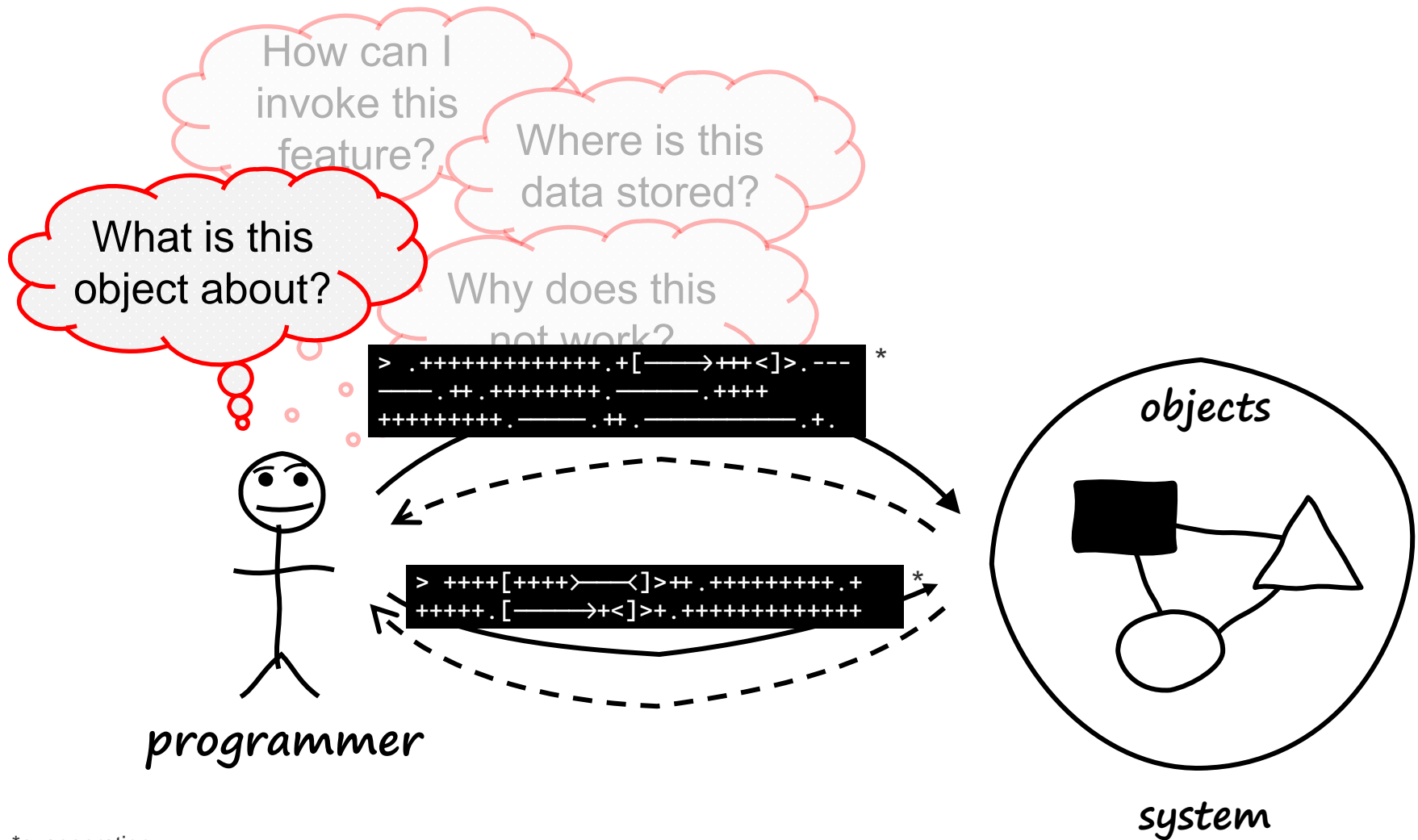
[TAE2022]

# Motivation



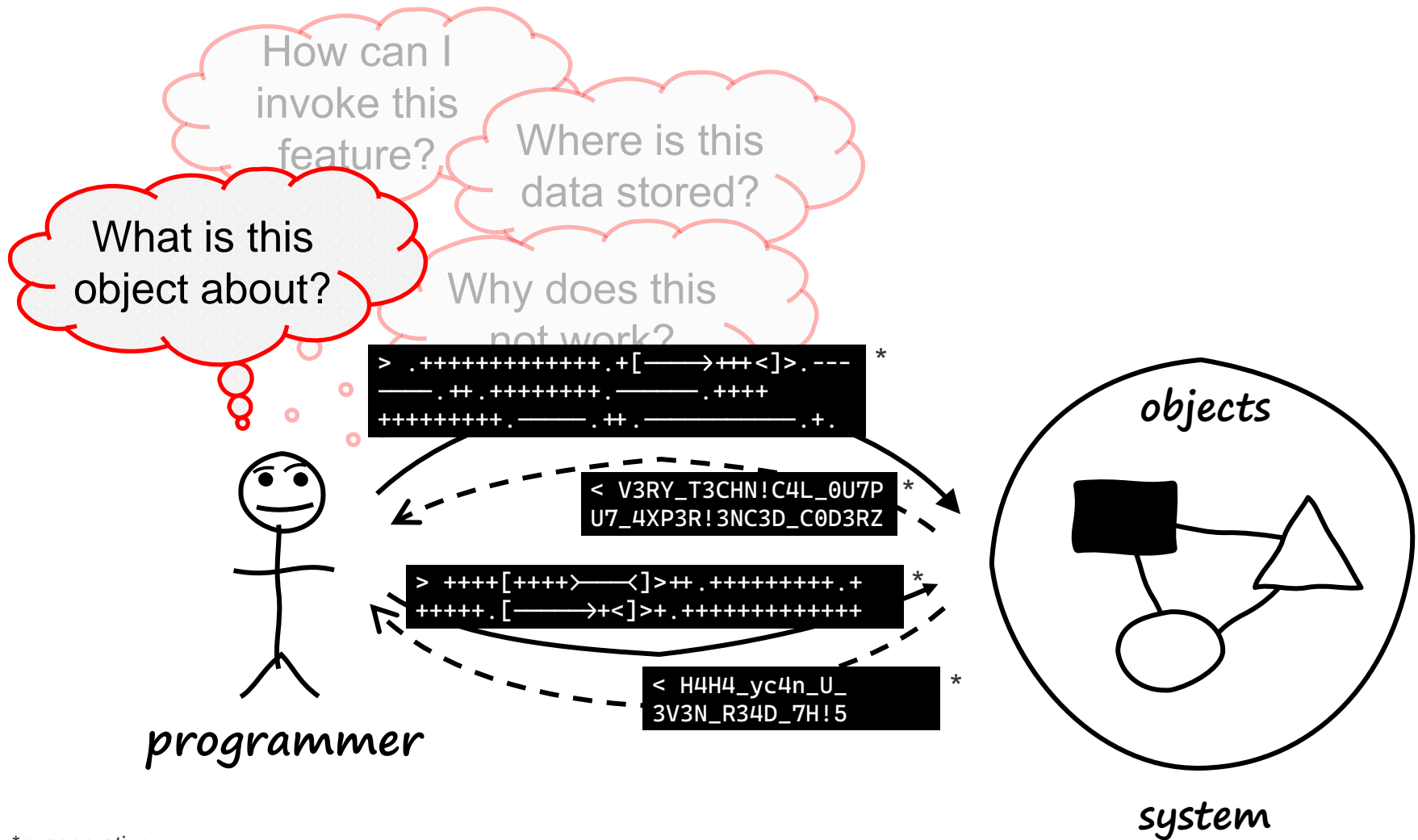
\*exaggeration

# Motivation



\*exaggeration

# Motivation



\*exaggeration

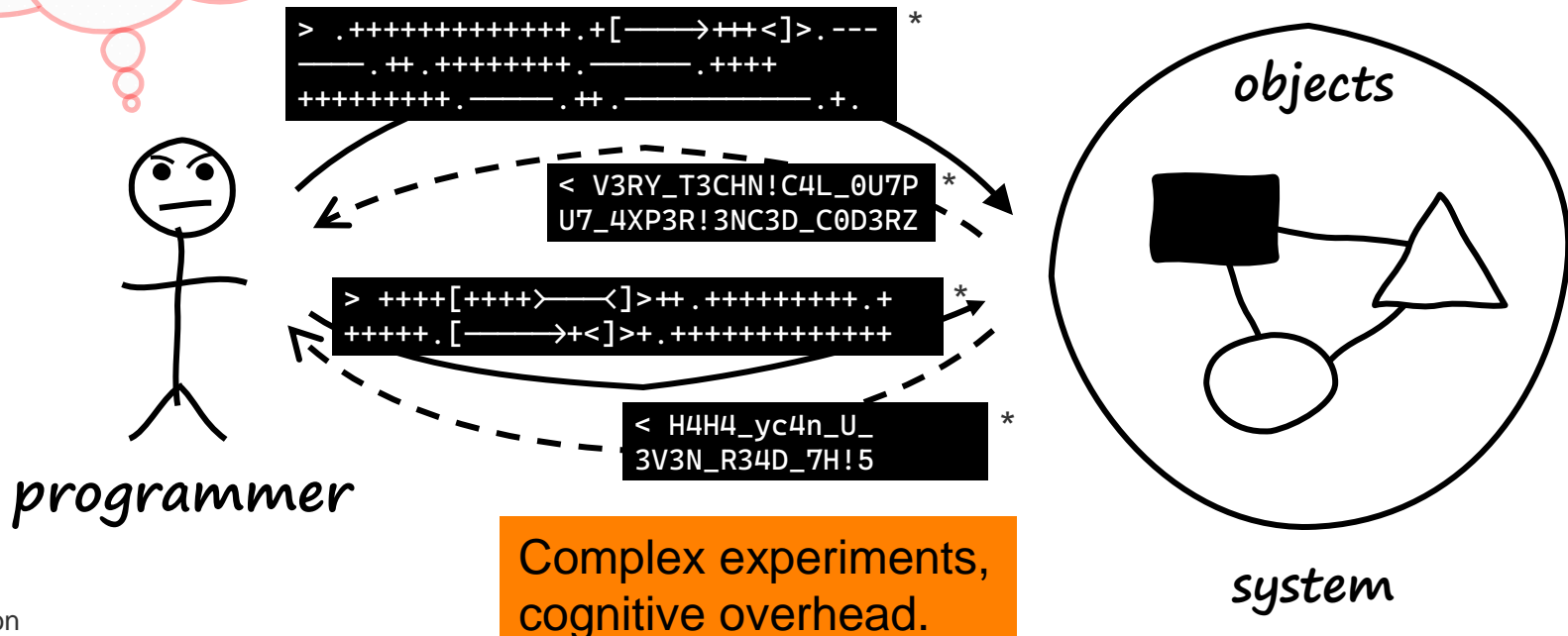




# Idea

- **Generative AI agents** already support programmers at different comprehension and interaction tasks ...
- Why not use them to **streamline** and **augment object exploration**?

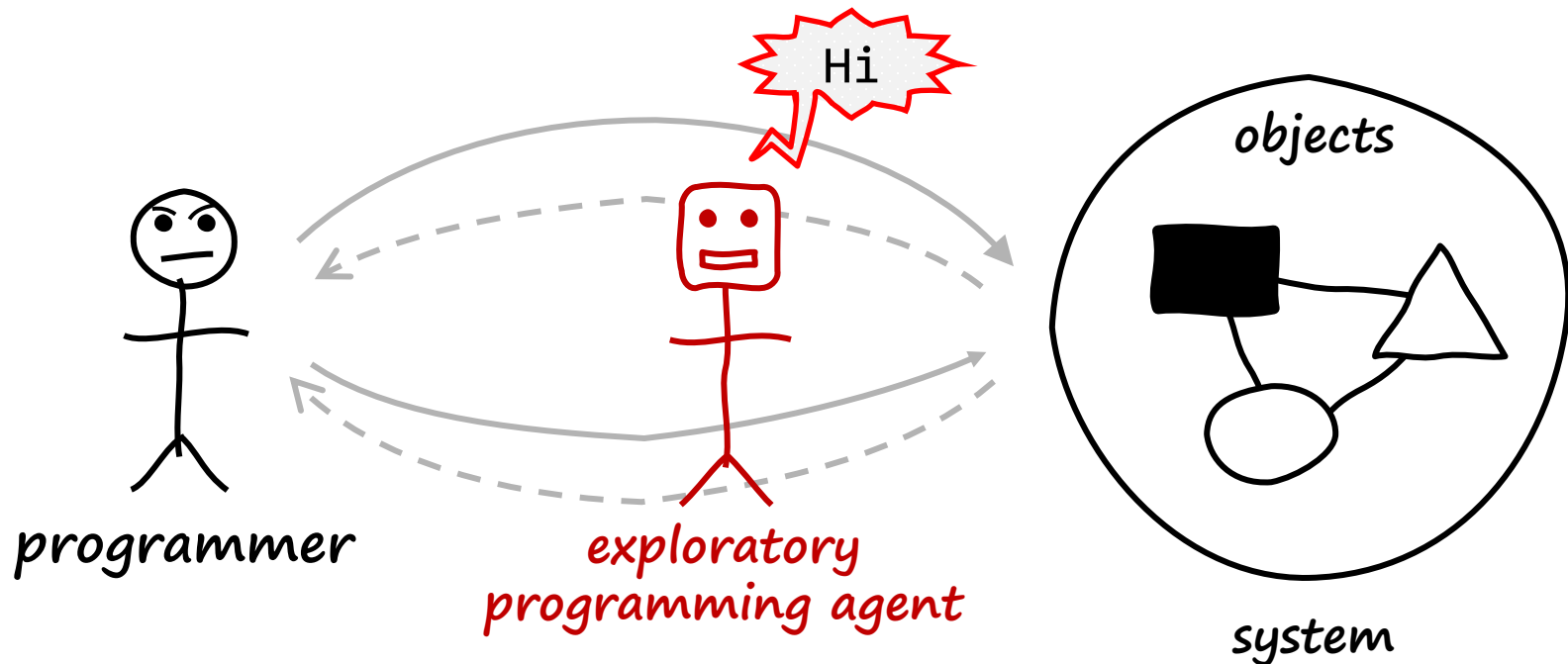
What is this object about?



\*exaggeration

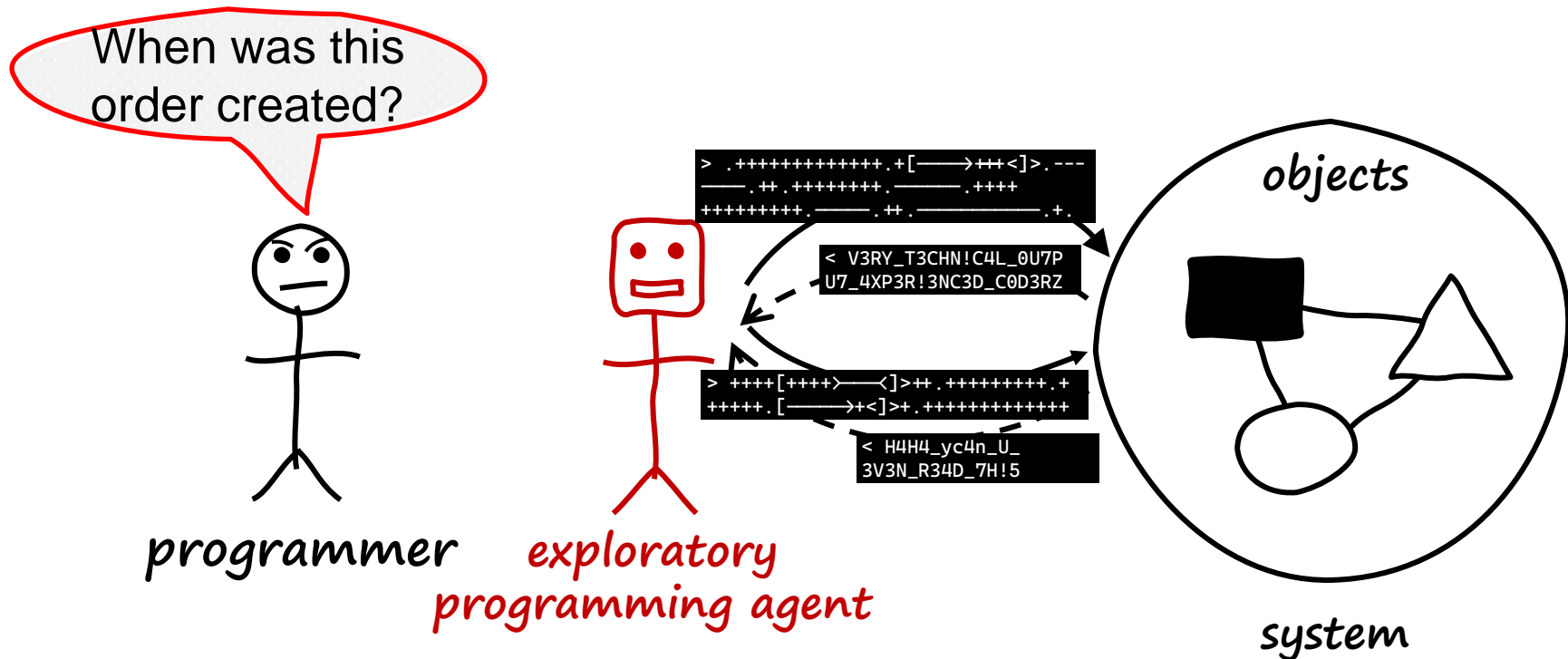
# Idea

- **Generative AI agents** already support programmers at different comprehension and interaction tasks ...
- Why not use them to **streamline** and **augment object exploration**?



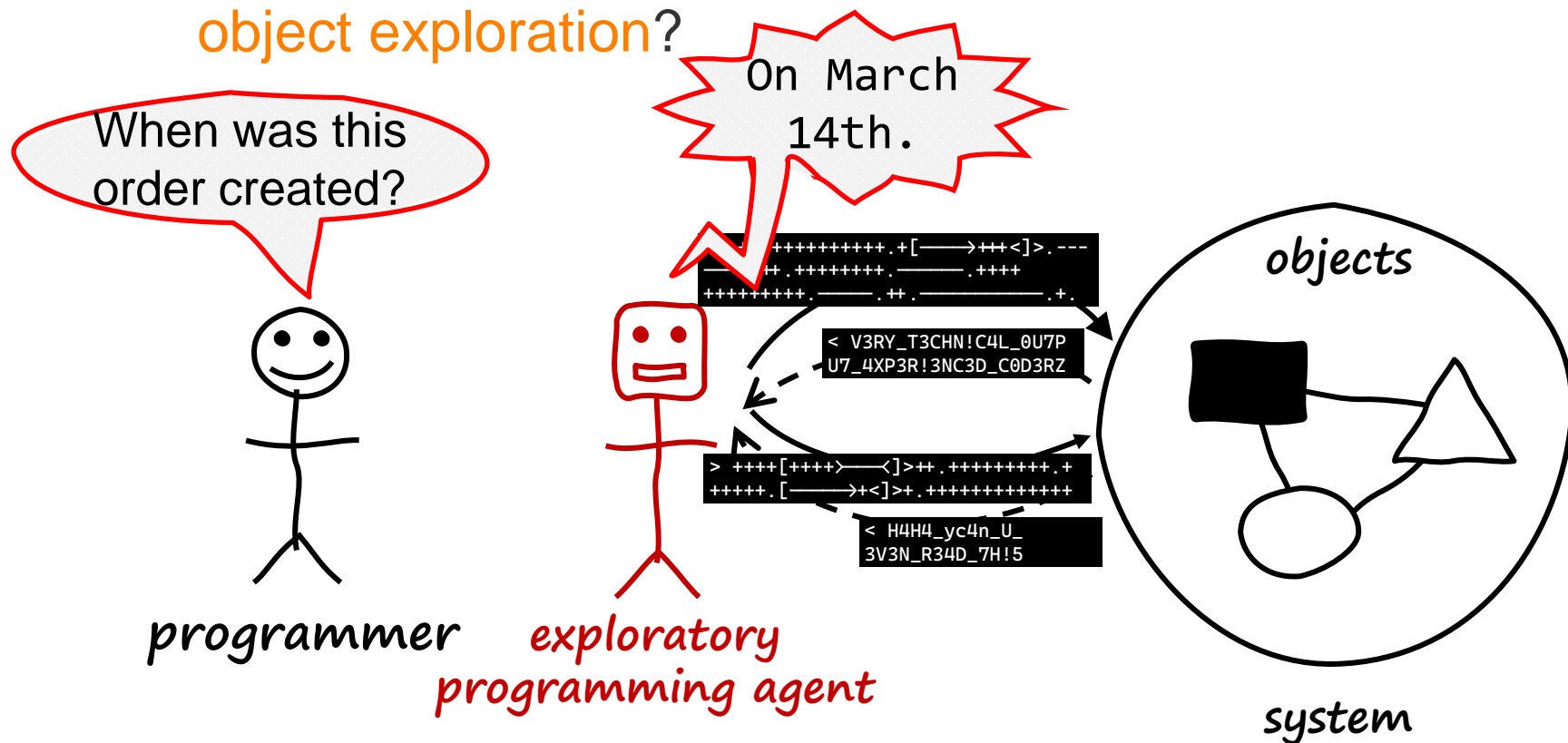
# Idea

- **Generative AI agents** already support programmers at different comprehension and interaction tasks ...
- Why not use them to **streamline** and **augment object exploration**?



# Idea

- **Generative AI agents** already support programmers at different comprehension and interaction tasks ...
- Why not use them to **streamline** and **augment object exploration**?



# Research Question

How can we support exploratory programming through **semantic object interfaces** that enable **contextual, natural-language conversations** with or about objects?

# Contributions



Semantic object  
interfaces  
framework



Prototype  
for Squeak/Smalltalk,  
using GPT-4o



Evaluation  
from our experience

# Background

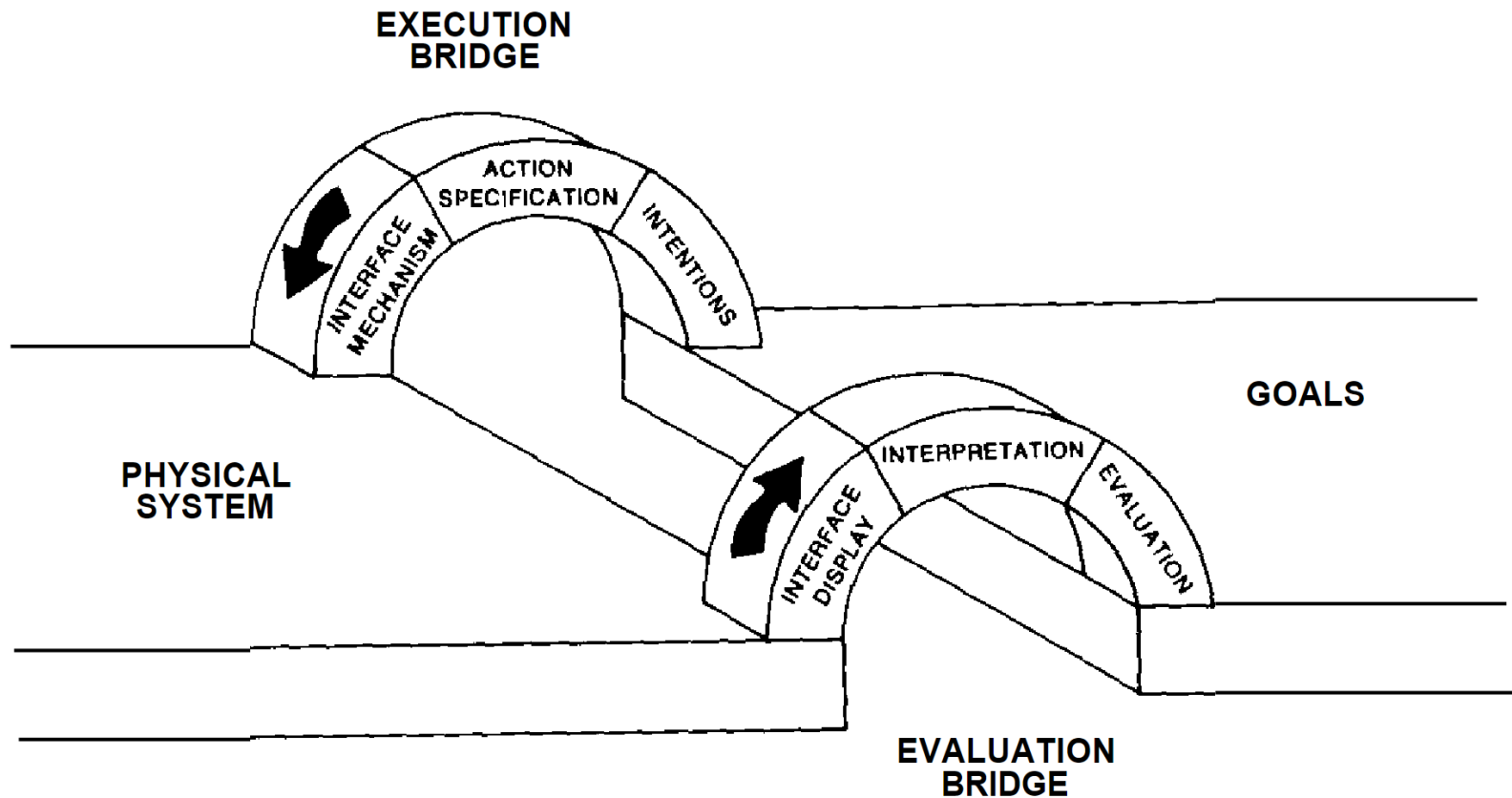
# Background: Exploratory Programming

- Exploratory programmers **understand** problems and systems **iteratively** [SAN1988,REI2019]
- They work like **researchers**: they formulate **questions**, conduct **experiments**, and **evaluate** results
- They conduct vivid and extensive **conversations** with systems through many small experiments [TAE2022]





# Background: Two Gulfs of HCI



[NOR1986]

# Background: Generative AI

```

1  module.exports = function allowAdminAccessMiddleware(req, res, next) {
2    if (req.user && req.user.admin) {
3      return next();
4    } else {
5      return res.status(401).send({
6        success: false,
7        message: 'Unauthorized',
8      });
9    };
10 };

```

Tabnine Pro

[Tabnine]

```

1  (defpackage :weather
2    (:use :cl :drakma)
3    (in-package :weather)
4
5    (defun get-weather (location)
6      "Fetch weather information for the given location from wttr.in."
7      (let ((url (format nil "https://wttr.in/~A?format=3" location)))
8        (with-output-to-string (stream)
9          (write-string (drakma:http-request url) stream))))
10
11     (defun display-weather (location)
12       "Fetch and print the weather information for a location."
13       (let ((weather (get-weather location)))
14         (format t "Weather for ~A: ~A~%" location weather)))
15
16     ;; Example usage:
17     (display-weather "Berlin")

```

[ChatGPT]

ChatGPT

Ensure that the 'location' is properly URL-encoded to avoid issues with special characters in the location name.

Apply

monalisa

How do I design a responsive layout in @primer?

**Copilot**

To design a responsive layout in primer, you should utilize the Primer design guidelines which provide common grammar and vocabulary to build cohesive, accessible, responsive, and efficient experiences. For responsive design considerations specifically, you can check out the section on responsive foundations. Primer provides two levels of abstraction for handling responsive designs: viewport ranges and breakpoints. Viewport ranges target common scenarios when designing responsive layouts, and breakpoints enable designers to fine-tune their responsive experiences.

Using 3 results from Primer docs

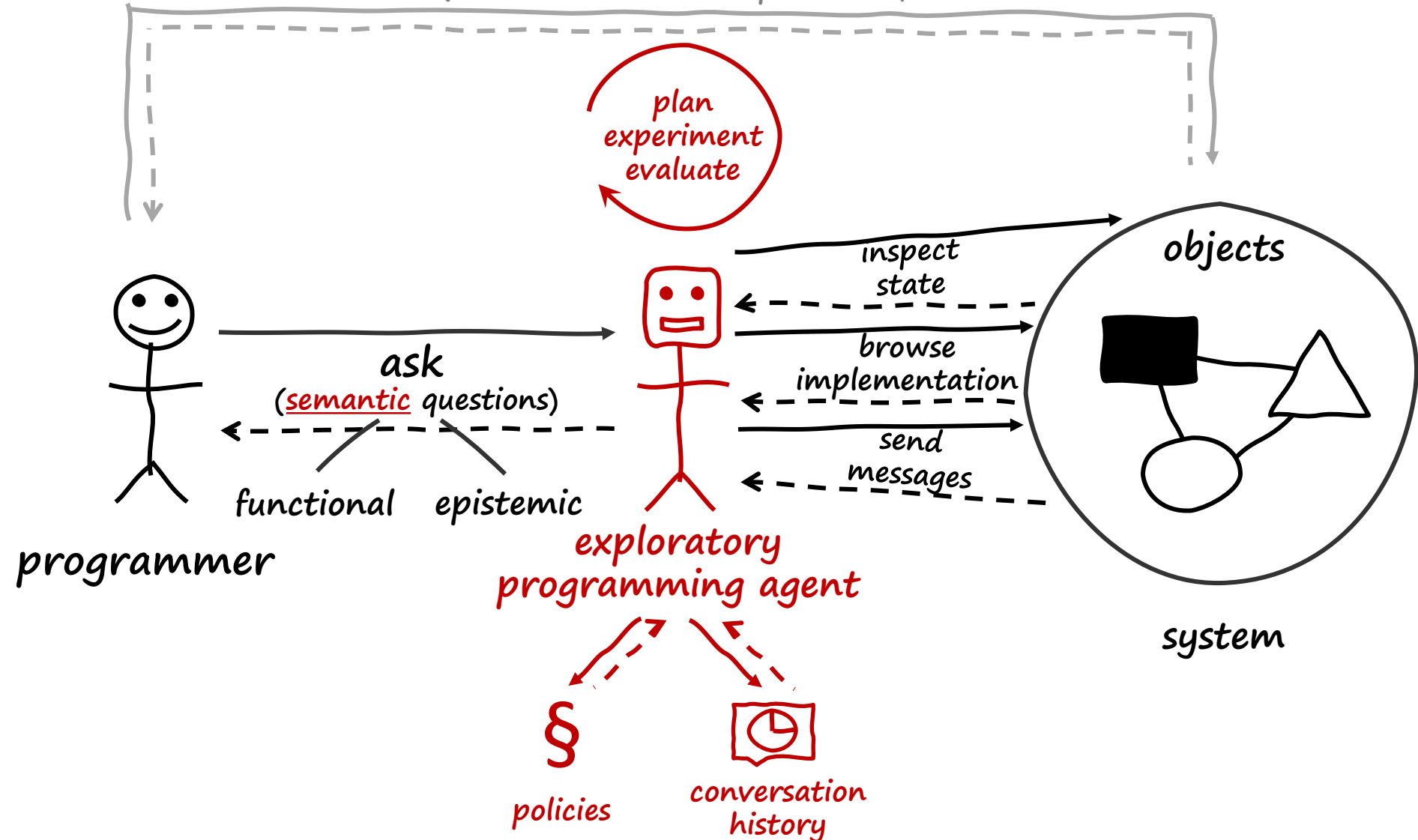
1. Viewport ranges
2. Introduction to Primer
3. Developing with Primer

Ask a question or type '/' for topics

[GitHub Copilot]

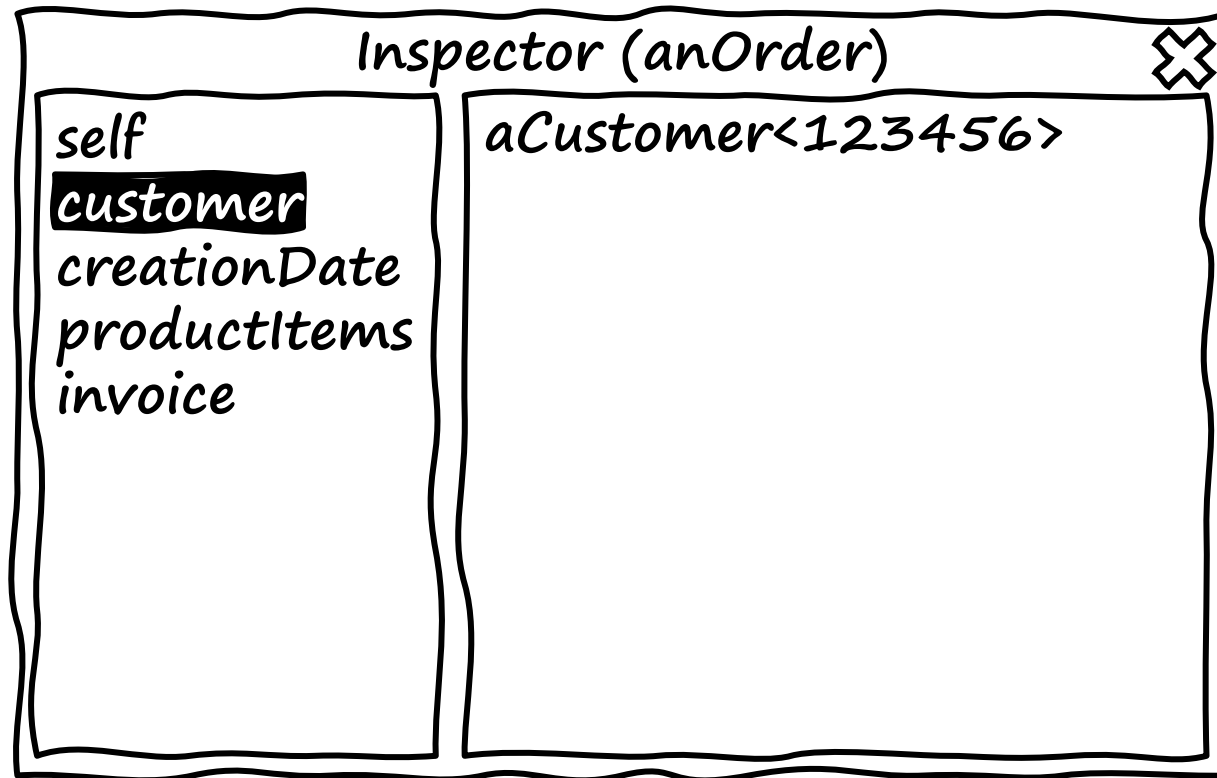
# Approach: Semantic Object Interfaces

(traditional manual experiments)

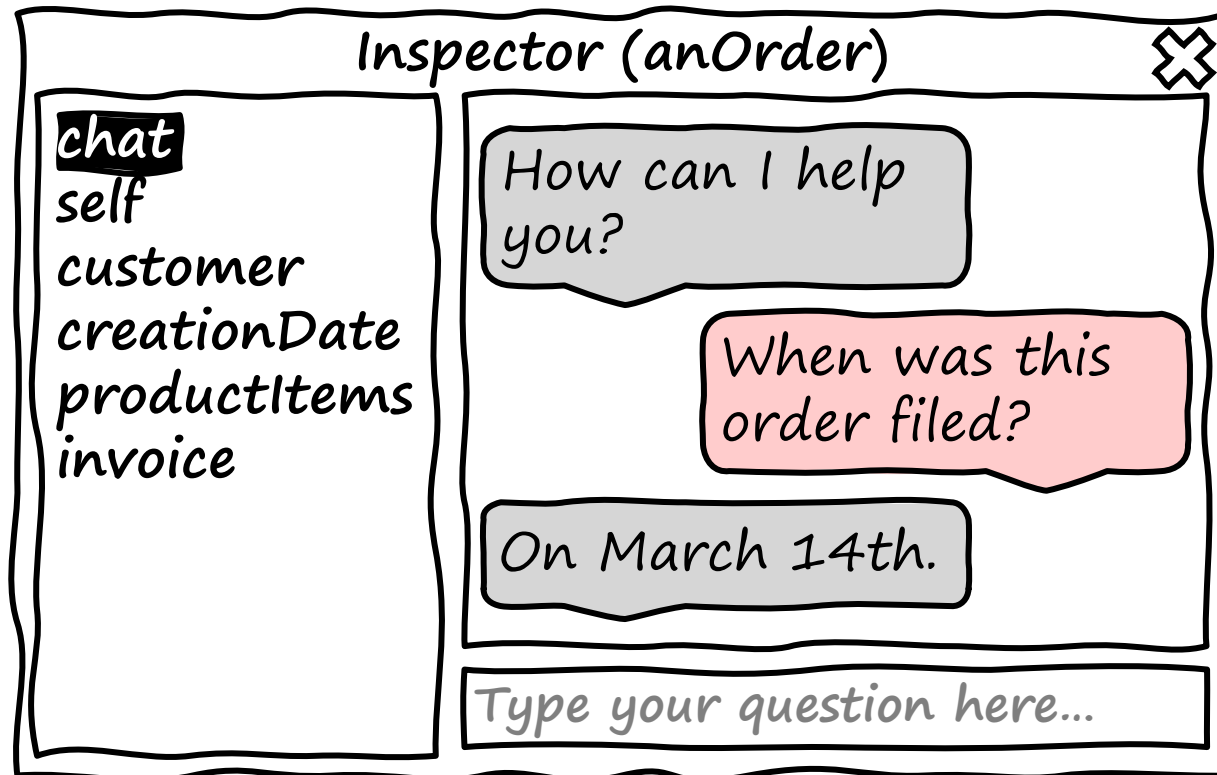


# Integration into Programming System

# Integration: Conversation Mode for Object Inspection Tools



# Integration: Conversation Mode for Object Inspection Tools



# Integration: Semantic Messaging for Scripts

- Traditional scripting:
  - `aProduct customer lastName.`
  - `(aProduct orderItems detectMax: #quantity) product.`
- Scripting with semantic messages:
  - `aProduct orderItems mostOftenBoughtOne.`
  - `aProduct mostPopularArticle.`
  - `aProduct numberOfSalesTo: aCustomer.`
  - `aProduct countSalesFrom: '2023Q3' to: '2023Q4'.`



# Building an Exploratory Programming Agent

A light red rectangular box with a red border, containing the text 'GPT-4o' in black font.

GPT-4o

# Building an Exploratory Programming Agent: Implementing Policies through Prompts

## HEADER

### *Exploratory programming agent*

- System:** You are an exploratory programming agent... • *identity*
- System:** You can call the following functions... • *interface description*
- System:** To solve a task, you should... • *rules and traits for problem solving*

### *Conversation mode (optional)*

- System:** You are an object... • *object identity*
- System:** Keep your answers brief... • *output format*

### *Semantic messaging (optional)*

- System:** You must call the `evalAndReturn` function... • *output format*
- System:** Format the return value as...

### *Bootstrapping the exploration*

- System:** This object represents... • *hardcoded semantic context*
- Assistant:** To understand this object, I will first... • *zero-shot chain-of-thought*
- Assistant:** `eval("self printString")`
- Result:** `an Object(12345)`
- Assistant:** `eval("self allInstVarNames")` • *initial object context*
- Result:** `#('foo' 'bar')`

## BODY

- User:** What does this object...? • *user question*

# Building an Exploratory Programming Agent: System Interfaces for Experiments

---

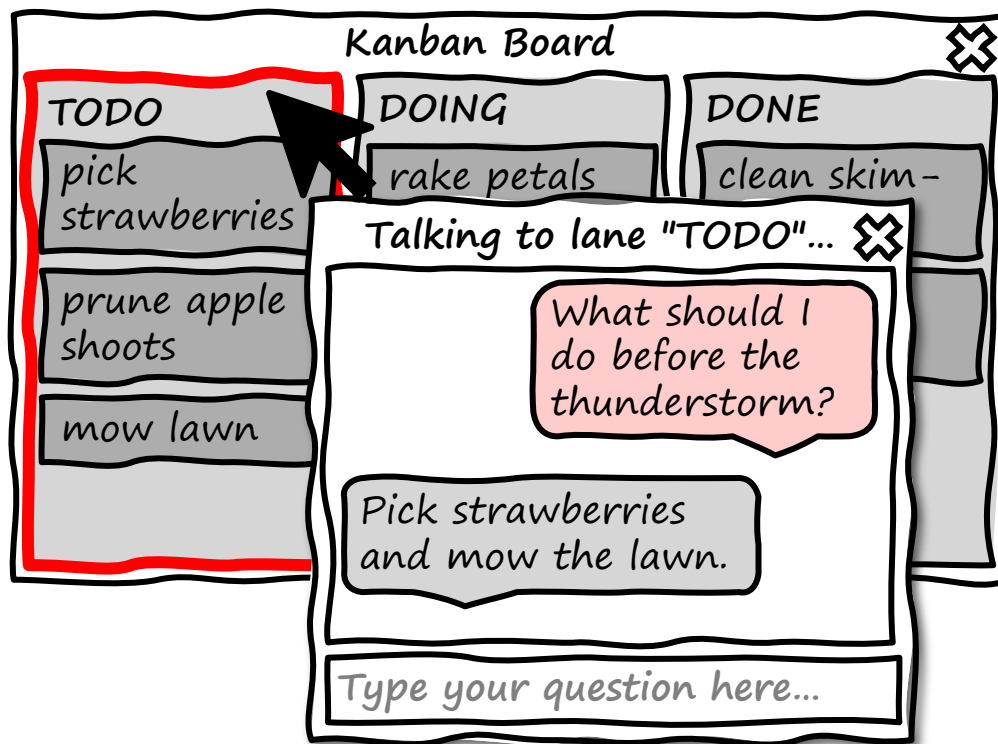
Function	Description
<b>eval(expression)</b> <i>Example:</i> <code>eval("self customer")</code>	Evaluate a Smalltalk expression in the context of the explored object and return the result or error. Can be executed in isolation.
<b>evalAndReturn(expression)</b>	Evaluate a Smalltalk expression in the context of the explored object and pass back the result to the sender of the original semantic message. Only available if the agent was invoked through a semantic message.
<b>browsePackage(packageName)</b>	Return a hierarchical list of classes within a package.
<b>browseClass(className)</b>	Enumerate all methods defined on a class or one of its superclasses or their metaclasses (for static methods), grouped by the defining class and the method category (protocol) within the class organization.
<b>browseMethod(className, selector)</b>	Retrieve the source code of a method defined in a class.
<b>browseSenders(selector[, query])</b> <i>Examples:</i> <code>browseSender("printOn:")</code> <code>browseSender("printOn:", "date yy-mm-dd")</code>	Search the system for all methods that send messages with the name of a selector and return a subset.

---

# Demo

# Toward a Semantic Toolset

- Idea: Allow users of **object-oriented user interfaces** to **talk to domain objects** on their screen



# Toward a Semantic Toolset

- Idea: Allow users of **object-oriented user interfaces** to **talk to domain objects** on their screen
- Many exploratory programming tools employ object-oriented interfaces:
  - Structural navigation tools (such as Smalltalk code browsers)
  - Projectional editors (based on AST)
  - Symbolic debuggers (based on process/call stack)
  - Profilers (based on trace)
  - ...

# Discussion

- Feasibility of the exploratory programming agent
- Programming experience of semantic object interfaces

# Discussion: Feasibility

- **Limitations of LLMs**



## Errors

Hallucinations, incorrect reasoning,  
invalid code



## Failures

Insufficient answers, endless  
trial & error, refused tasks



### Train specific abilities?

- Proficiency with Squeak/Smalltalk language + frameworks
- Exploratory practice

- **Performance**

	Response times	Monetary cost	
<b>Simple tasks</b>	2 s – 4 s	\$0.10 – \$0.50	\$1 – \$60 per hour [KUB2018]
<b>Complex tasks</b>	5 s – 15 s	\$0.5 – \$5	



**Fine-tuned or small language models?** [MAG2023]  
**Optimize prompts?**



# Discussion: Programming Experience

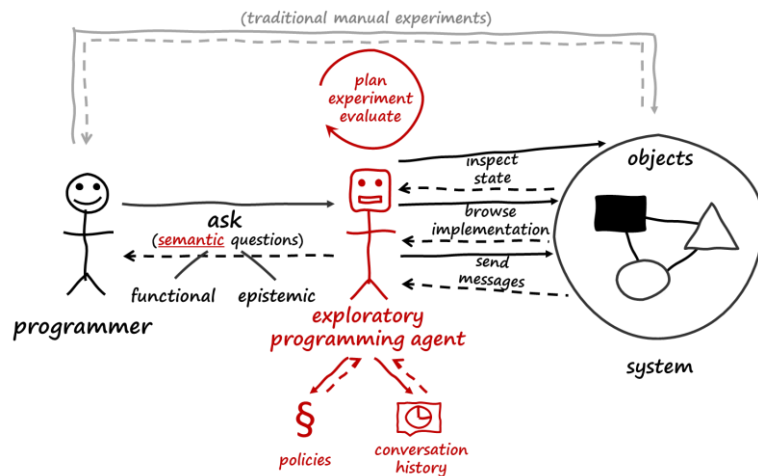
- Higher **abstraction level**
  - + Reduced interruptions and cognitive overhead
  - + Improved semantic immediacy and flow [CSI2008]
  - Leaky abstractions [SPO2004]
  - Missed serendipitous discoveries
- **Natural-language** interface
  - + Reduced gulf of execution and evaluation
  - + Implicit context
- Current LLMs
  - Limited trust
  - Temporal distances for complex questions
  - Fear of expenses
  - Energy & water consumption [LI2023]

# Future Work

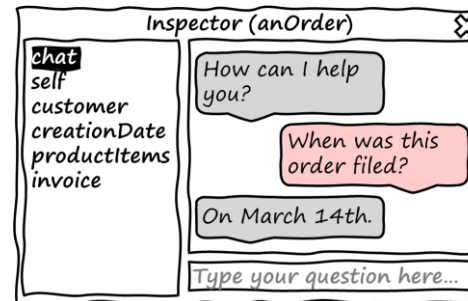
- How can we improve the **capability** and **performance** of the **exploratory programming agent**?
  - ... by fine-tuning a smaller language model?
  - ... by optimizing prompts and function interfaces?
- How can we improve the **collaboration** of programmers and exploratory agents? [THI2024]
  - ... by collecting further **implicit context** for agents?
  - ... by improving the **explanation** of semantic answers?
  - ... by **separating responsibilities** between programmers and agents more clearly?

# Conclusion

## Semantic object interfaces



## Integrations into programming systems



```
aProduct orderItems mostOftenBoughtOne.
aProduct numberOfSalesTo: aCustomer.
aProduct countSalesFrom: '2023Q3' to:
'2023Q4'.
```

## Findings

- Semantic object interfaces improve **semantic experience**, reduce **cognitive overhead**, and avoid **distractions**
- Programmers delegate **control** and miss **serendipitous discoveries**
- Need to **optimize** and **fine-tune** LLMs for exploratory programming

## Applications

```
Text
chat
self
all inst vars
string
runs
1
2
3
4
5
User: what attributes are in this text
Assistant: The text contains the following attributes:
- TextEmphasis with code 1
- An empty attribute set
- TextEmphasis with code 2
User: what do the codes mean
Assistant: The codes for TextEmphasis mean the following:
- 1: bold
- 2: italic
- 4: underlined
- 8: narrow
- 16: struck out
User: multiple different ways to make self all italic? answer only code!
Assistant:
- self addAttribute: TextEmphasis italic
- self addAttribute: TextEmphasis italic from: 1 to: self size
- Text string: self string attribute: TextEmphasis italic
User:
Evaluate expressions on inspected object explore
```

```
Halt:
UndefinedObject(Object)->halt
UndefinedObject->Dolt
Compiler->evaluateCue:ifFail:logged:
[] in SmalltalkEditor(TextEditor)->evaluateSelectionAndDo:
FullBlockClosure(BlockClosure)->onDo:
SmalltalkEditor(TextEditor)->evaluateSelectionAndDo:
SmalltalkEditor(TextEditor)->evaluateSelection
SmalltalkEditor(TextEditor)->doIt
Proceed Restart Into Over Through Full Stack Where Trace It
evaluateCue: aCue ifFail: failBlock logged: logFlag
*Compiles the cue source into a parse tree, then generates code into a method.
Finally, the compiled method is invoked from here via Args.executeMethod, hence
the system no longer creates DoIt method litter on errors.*
[methodNode method value]
methodNode == self compileCue: aCue noPattern: true ifFail: { failBlock value}.
chat
self
all inst vars
parser
cue
a Compiler
chat
thisContext
all temp vars
aCue
failBlock
logFlag
methodNode
method value
User: where is failBlock from and what does it do
Assistant: The failBlock originates from SmalltalkEditor->evaluateSelectionAndDo: and is designed to flash the text editor's selection area, then return nil if the evaluation fails.
User:
```

Thank you!



## Further Information

- Christoph Thiede, Marcel Taeumel, Lukas Böhme, and Robert Hirschfeld. **Talking to Objects in Natural Language: Toward Semantic Tools for Exploratory Programming.** In: *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '24)*, October 23–25, 2024, Pasadena, CA, USA. ACM, New York, NY, USA, 17 pages.  
<https://doi.org/10.1145/3689492.3690049>
- Poster: <https://linqlover.github.io/LinqLover/posters/Onward24%20Talking%20to%20Objects.pdf>
- Artifacts:
  - <https://github.com/hpi-swa-lab/SemanticSqueak>
  - <https://github.com/hpi-swa-lab/Squeak-SemanticText>

# Literature

- [BAR2023] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1, Article 78 (4 2023), 27 pages. <https://doi.org/10.1145/3586030>
- [BEC2020] Tom Beckmann, Stefan Ramson, Patrick Rein, and Robert Hirschfeld. 2020. Visual Design for a Tree-Oriented Projectual Editor. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming* (Porto, Portugal) (<Programming> '20 Companion). ACM, New York, NY, USA, 113–119. <https://doi.org/10.1145/3397537.3397560>
- [CSI2008] Mihaly Csikszentmihalyi. 2008. *Flow: The Psychology of Optimal Experience* (1 ed.). Harper Perennial, New York. ISBN: 978-0-06-133920-2
- [KUB2018] Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2018. The Road to Live Programming: Insights from the Practice. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) (ICSE '18). ACM, New York, NY, USA, 1090–1101. <https://doi.org/10.1145/3180155.3180200>
- [NOR1986] Donald A. Norman. 1986. *Cognitive Engineering*. Lawrence Erlbaum, Hillsdale, NJ, USA, 31–61.
- [REI2019] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2019. Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness. *The Art, Science, and Engineering of Programming* 3, 1 (07 2019), 33 pages. <https://doi.org/10.22152/programming-journal.org/2019/3/1>
- [SAN1988] David W. Sandberg. 1988. Smalltalk and Exploratory Programming. *SIGPLAN Not.* 23, 10 (1988), 85–92. <https://doi.org/10.1145/51607.51614>
- [SPO2004] Joel Spolsky. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity*. Berkeley, CA, USA: Apress, 2004. ISBN: 978-1-59059-389-9
- [TAE2022] Marcel Taeumel, Jens Lincke, Patrick Rein, and Robert Hirschfeld. 2022. A Pattern Language of an Exploratory Programming Workspace. In *Design Thinking Research: Achieving Real Innovation*, Christoph Meinel and Larry Leifer (Eds.). Springer, Cham, 111–145. [https://doi.org/10.1007/978-3-031-09297-8\\_7](https://doi.org/10.1007/978-3-031-09297-8_7)
- [THI2024] Christoph Thiede. 2024. *The Semantic Workspace: Augmenting Exploratory Programming with Integrated Generative AI Tools*. Master Thesis (to be defended). Hasso Plattner Institute, University of Potsdam. <https://github.com/LinqLover/semexp-thesis/releases/download/submission/semexp-thesis-oneside.pdf>
- [UNG1997] David Ungar, Henry Lieberman, and Christopher Fry. 1997. Debugging and the Experience of Immediacy. *Communications of the ACM* 40, 4 (apr 1997), 38–43. <https://doi.org/10.1145/248448.248457>

# Literature (AI)

- [CHE2021] Mark Chen et al. 2021. *Evaluating Large Language Models Trained on Code*. arXiv:[2107.03374](https://arxiv.org/abs/2107.03374) [cs.LG]
- [LEW2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (Vancouver, Canada) (NIPS '20, Vol. 33)*, Hugo Larochelle et al. (Eds.). Curran, RedHook, NY, USA, Article 793, 16 pages. ISBN: 978-1-7138-2954-6. arXiv:[2005.11401](https://arxiv.org/abs/2005.11401) [cs.CL]
- [LI2023] Pengfei Li, Jianyi Yang, Mohammad A. Islam, and Shaolei Ren. 2023. Making AI Less “Thirsty”: Uncovering and Addressing the Secret Water Footprint of AI Models. arXiv:[2304.03271](https://arxiv.org/abs/2304.03271) [cs.LG]
- [MAG2023] Lucie Charlotte Magister et al. 2023. Teaching Small Language Models to Reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.), Vol. 2: Short Papers. Association for Computational Linguistics, Toronto, Canada, 1773–1781. <https://doi.org/10.18653/v1/2023.acl-short.1511>
- [RAD2018] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. *Improving Language Understanding by Generative Pre-Training*. <https://openai.com/research/language-unsupervised>
- [WAY2023] Wayne Xin Zhao et al. 2023. *A Survey of Large Language Models*. 124 pages. arXiv:[2303.18223](https://arxiv.org/abs/2303.18223) [cs.CL]
- [WHI2023] Jules White et al. 2023. *A Prompt Pattern Catalog to Enhance PromptEngineering with ChatGPT*. 19 pages. [arXiv:2302.11382](https://arxiv.org/abs/2302.11382) [cs.SE]