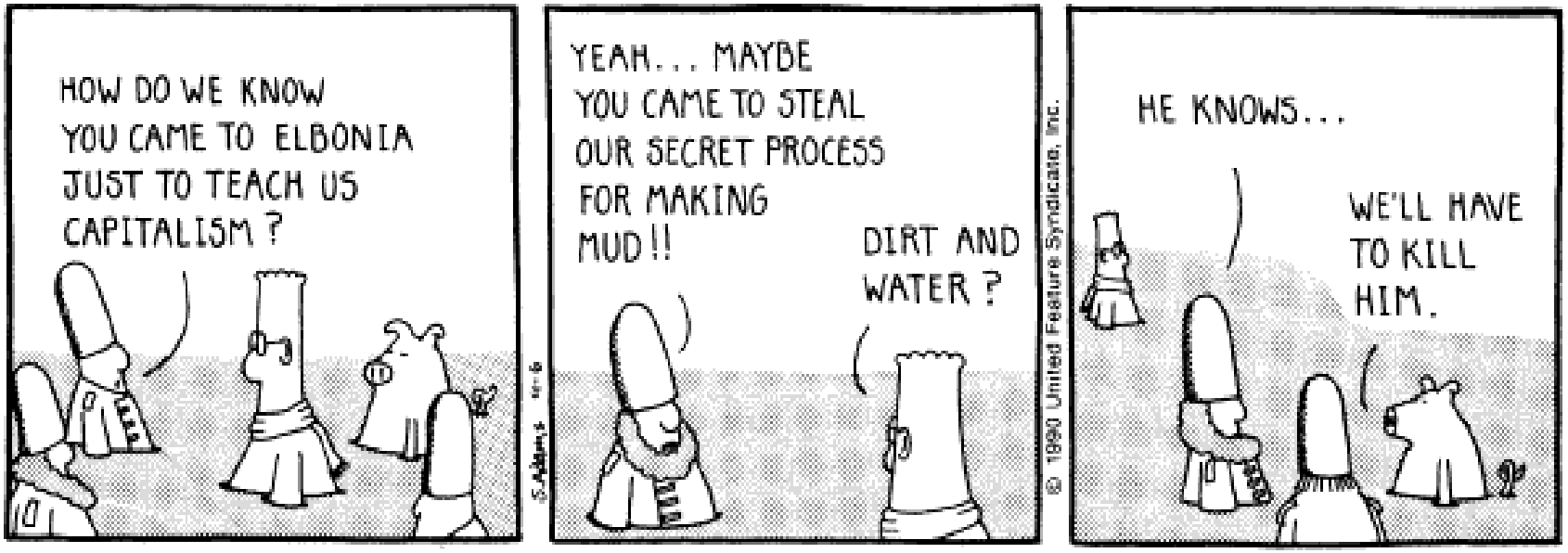
Reverse Engineering
Organization and Topics

Hasso-Plattner-Institut Potsdam
Software Architecture Group
http://www.hpi.uni-potsdam.de/swa/

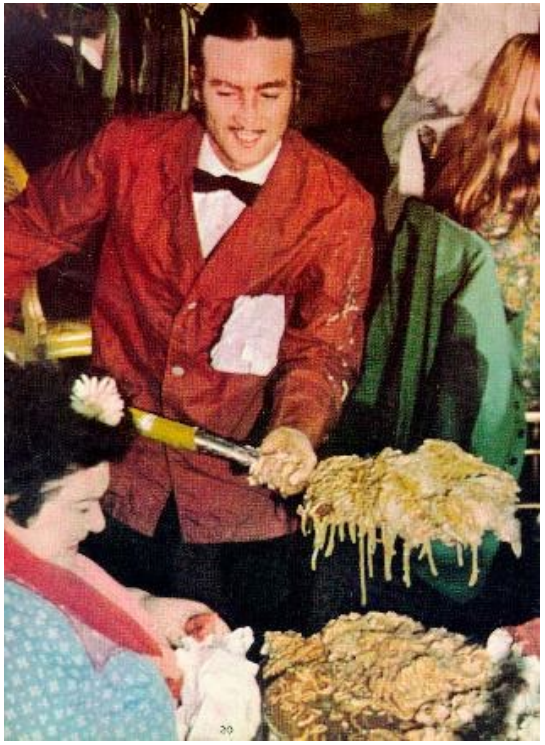
2018


HOW DO WE KNOW YOU CAME TO ELBONIA JUST TO TEACH US CAPITALISM ?
YEAH... MAYBE YOU CAME TO STEAL OUR SECRET PROCESS FOR MAKING MUD !!
DIRT AND WATER ?
© 1990 United Feature Syndicate, Inc.
HE KNOWS...
WE'LL HAVE TO KILL HIM.

Remember “Big Ball of Mud” SWT1?

“A **BIG BALL OF MUD** is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle. We’ve all seen them. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated. The overall structure of the system may never have been well defined. If it was, it may have eroded beyond recognition. Programmers with a shred of architectural sensibility shun these quagmires. Only those who are unconcerned about architecture, and, perhaps, are comfortable with the inertia of the day-to-day chore of patching the holes in these failing dikes, are content to work on such systems.”

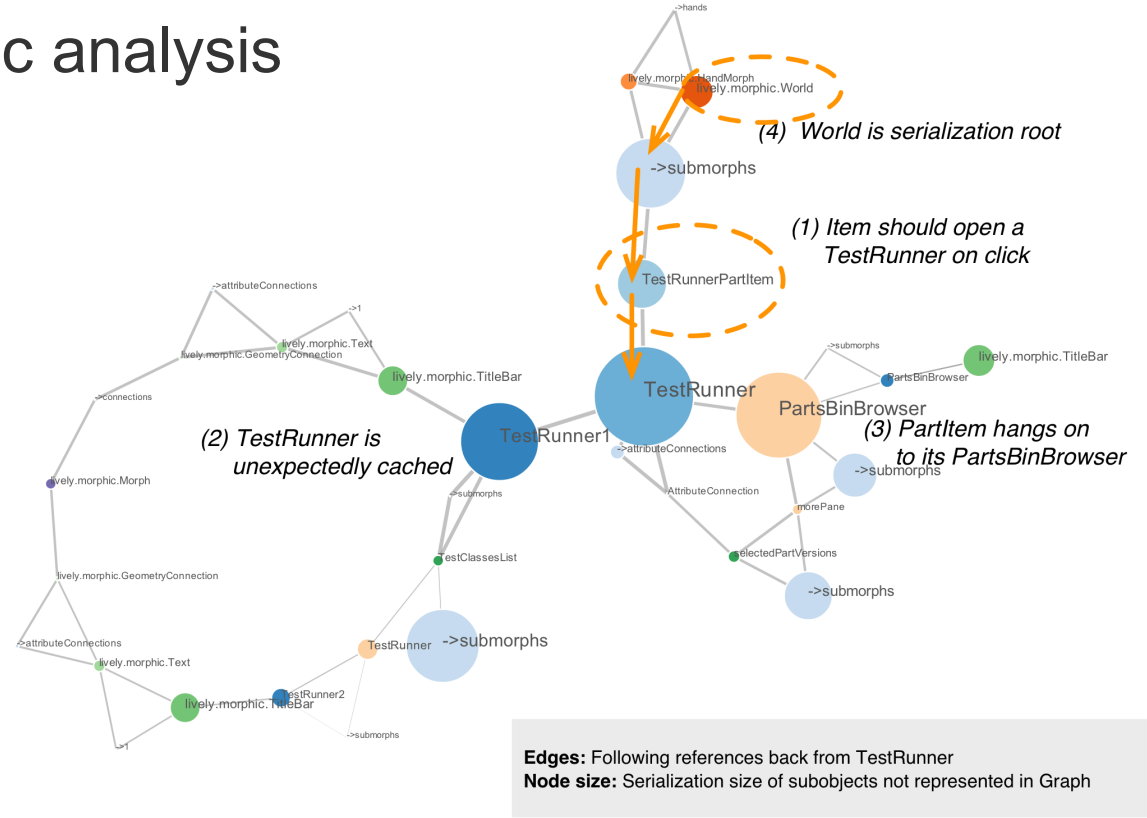
[Foote and Yoder 1997 Big Ball of Mud]



Lets do something about it…

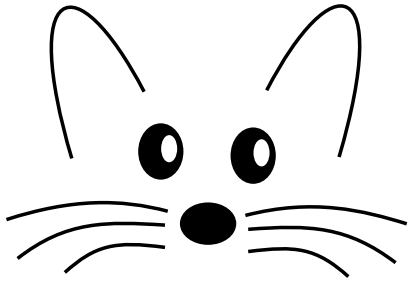
Introduction: Reverse Engineering

- Find out how something works, in order to …
- Change, extend, refactor, rebuild, use …
- Self-supporting Development Environments
- Static vs. dynamic analysis

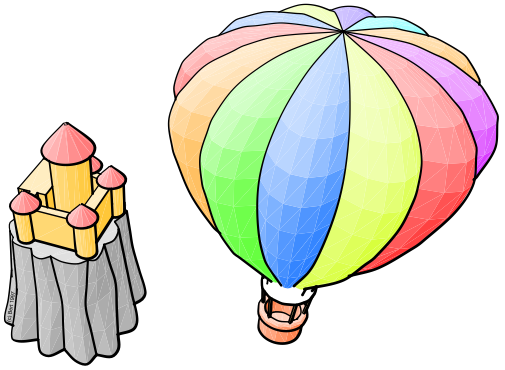


Development Environments

Lively Kernel, JavaScript,
WebComponents, D3

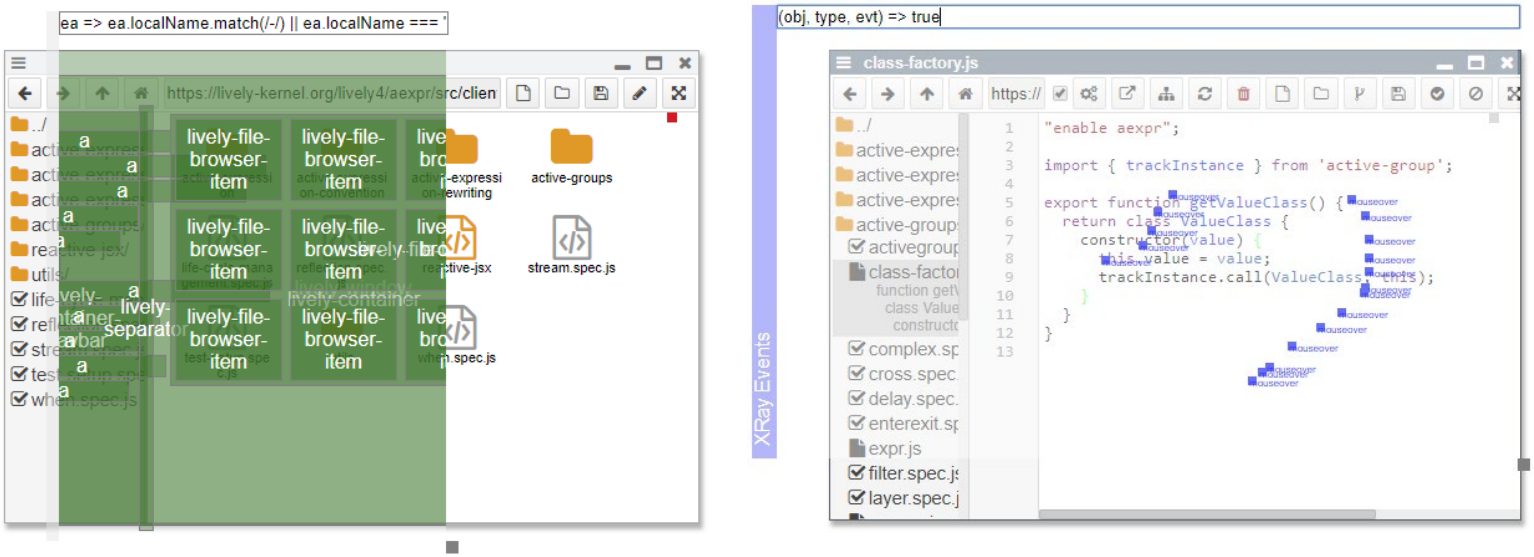


Squeak Smalltalk



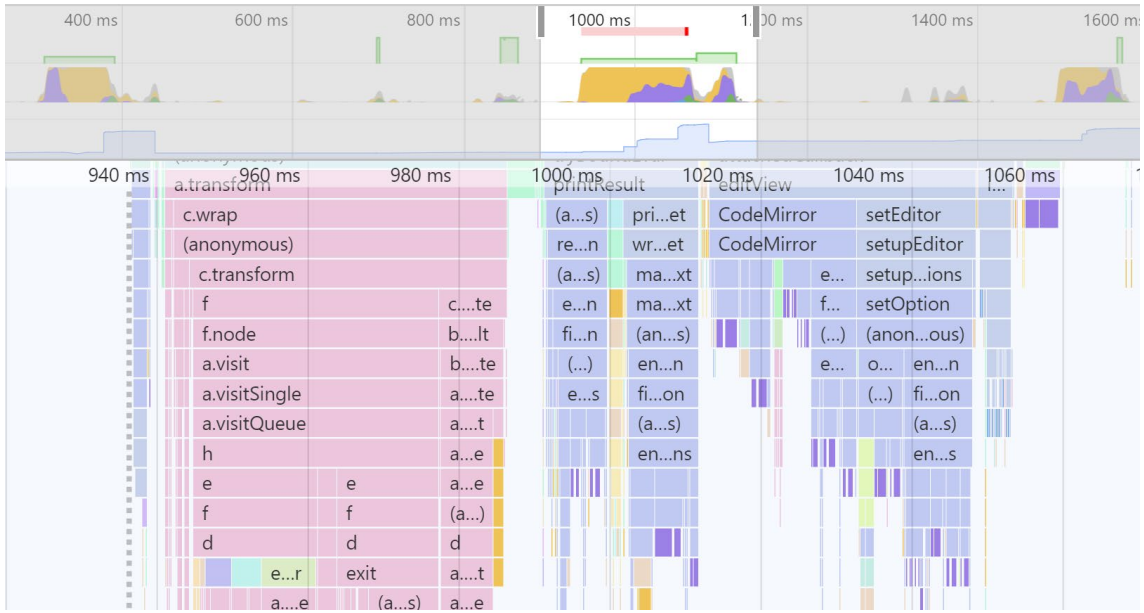
TOPICS

Xray through UI



- Lively XRay tools overlay graphical user interface to:
  - Reveal and inspect structure of the DOM hierarchy
  - Make user events visible
- **Goal:** Extend XRay tools to reveal behavior
  - Trace and show computation within components
  - Link computation to dynamic DOM structure and static source code
  - Highlight reactive components and their over-time behavior

Scriptable Trace Viewer in Lively4



Stack Trace in Blink DevTools

- Stack traces provide over-time view on system execution
- Generic tracer available as part of Blink DevTools, however, no domain knowledge or high-level abstractions are incorporated into such views
- **Goal:** Implement a scriptable trace viewer
  - Annotate and enhance with domain knowledge
  - Compute on and query traces

Tracking JS Built-ins for Changes

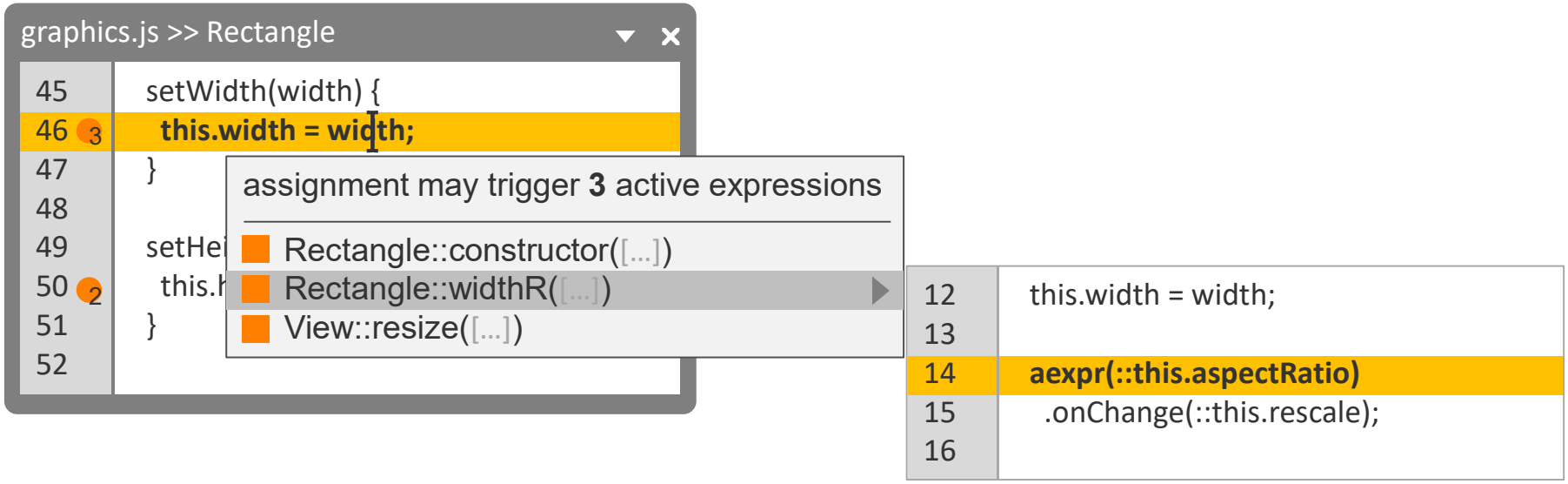
```
person.fullname() // Foo Bar

aexpr(() => person.fullname())
  .onChange(console.log)

person.firstname = `Hello` // prints `Hello Bar`
person.lastname = `World` // prints `Hello World`
```

- Active Exressions track arbitrary JavaScript expressions for change
- Using source code transformation, we intercept get and set operations to object members, locals, and globals
- Certain JavaScript built-ins do not follow this pattern, thus, cannot be tracked for changes currently

```
aexpr(() => input.value) ?
```

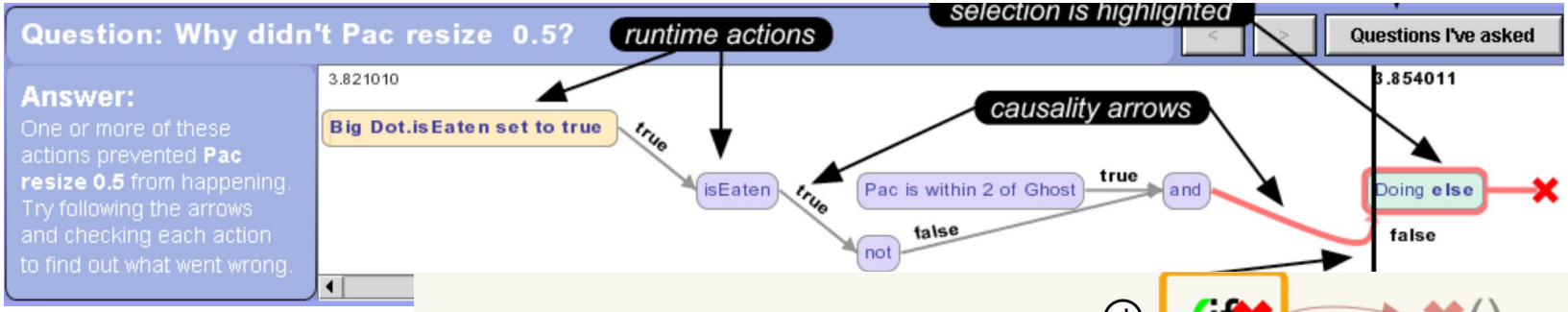
- Some DOM elements change properties under the hood
- **Goal:** Reify changes to JavaScript Built-ins and DOM elements
  - Integrate those mechanisms into Active Expressions

Static Analysis for High-Level Language Features

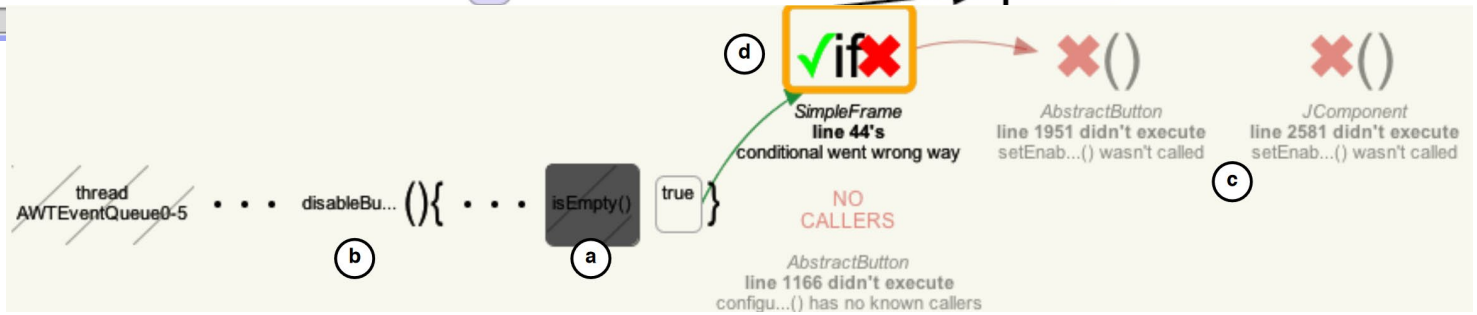


- Static analysis models like data dependency and call graphs aid semantic code navigation
- Dynamic language features complicate graph generation
- **Goal:** Extend a static analyzer to incorporate reactive programming concepts (such as active expressions)

Why and Why Not Questions



Alice WhyLine[1]



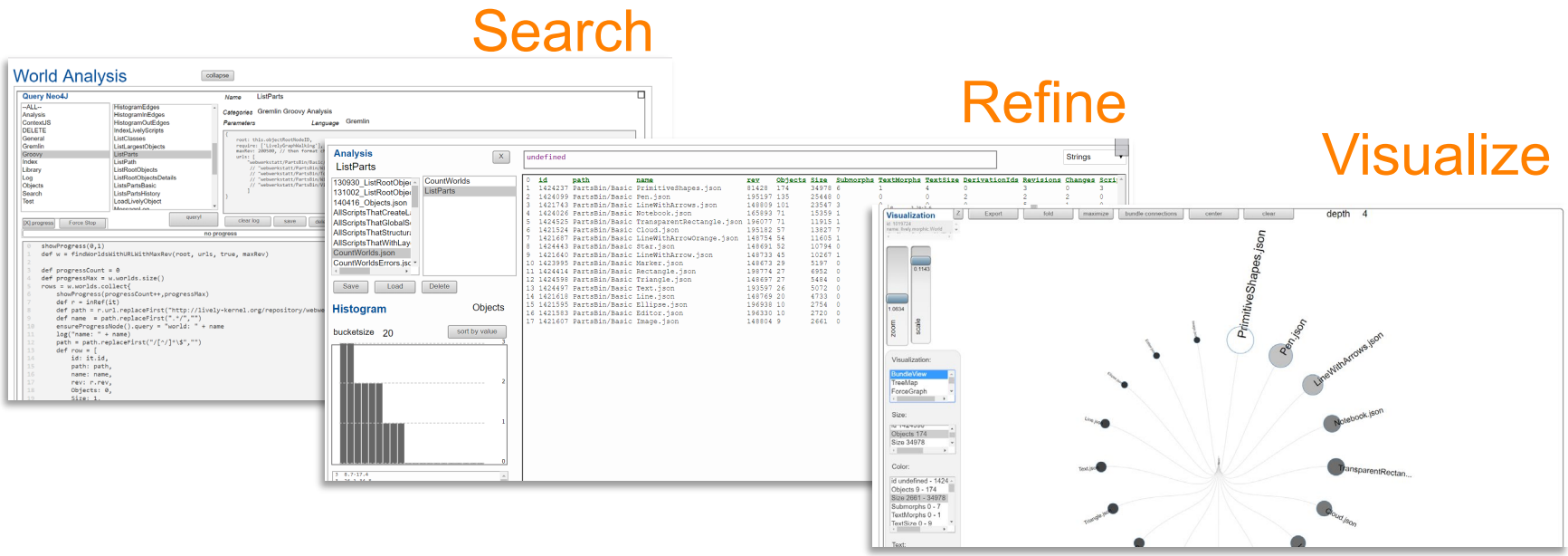
WhyLine in Java [2]

- Identifying the root cause of a failure involves an enormonous search space, often done manually
- The WhyLine automates this process, by allowing programmers to as why and why not questions about a program execution
- **Goal:** Implement a WhyLine for JavaScript

[1] Andrew J. Ko, and Brad A. Myers. “Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior” SIGCHI (2004), ACM.

[2] Andrew J. Ko, and Brad A. Myers. "Finding causes of program output with the Java Whyline" SIGCHI (2009), ACM.

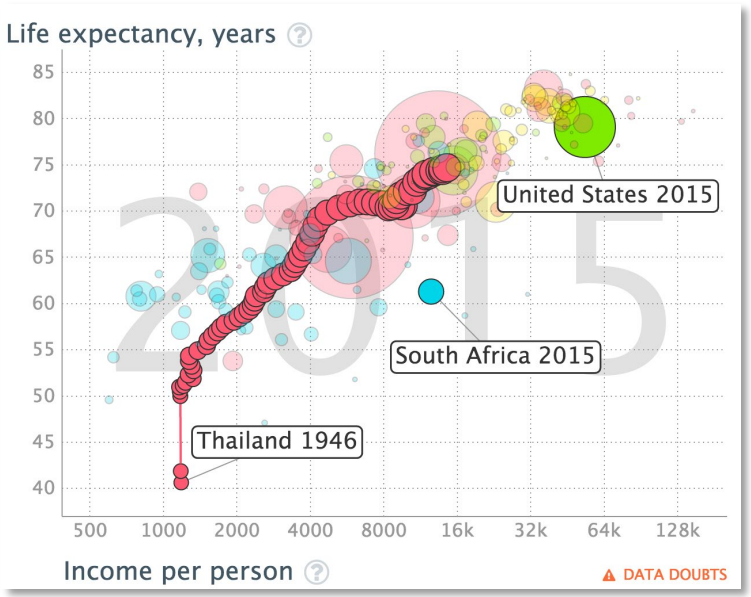
Github Explorer



- GitHub as powerful data source for all kinds of software architecture and engineering questions
- **Goal:** Provide a tool to explore GitHub data in Lively4
  - Stream data for immediate feedback
  - User experience similar to the Lively Kernel World Explorer (explorable schemata, tangible results)

Stroposcoping for non-graphical Objects

Stroposcopic View on Game Entities [1]



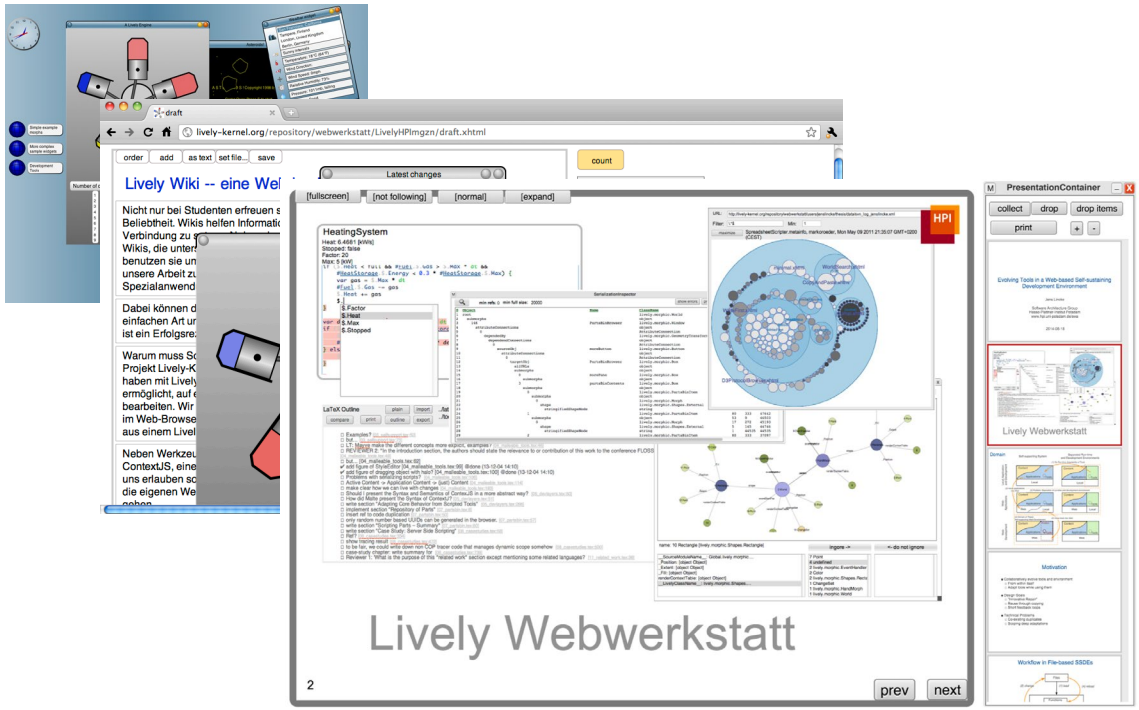
Rendering Multivariate Data over Time [2]

- Stroposcopic views allow to comprehend the over-time behavior of objects, but are limited to graphical objects
- Challenge: Arbitrary objects are n-dimensional data points
- Hypothesis: Only few properties of an object change frequently
- **Goal:** Provide a stroposcopic view for arbitrary JavaScript objects

[1] Bret Victor - Inventing on Principle [https://www.youtube.com/watch?v=PUv66718DII]
[2] Gapminder [https://www.gapminder.org/]

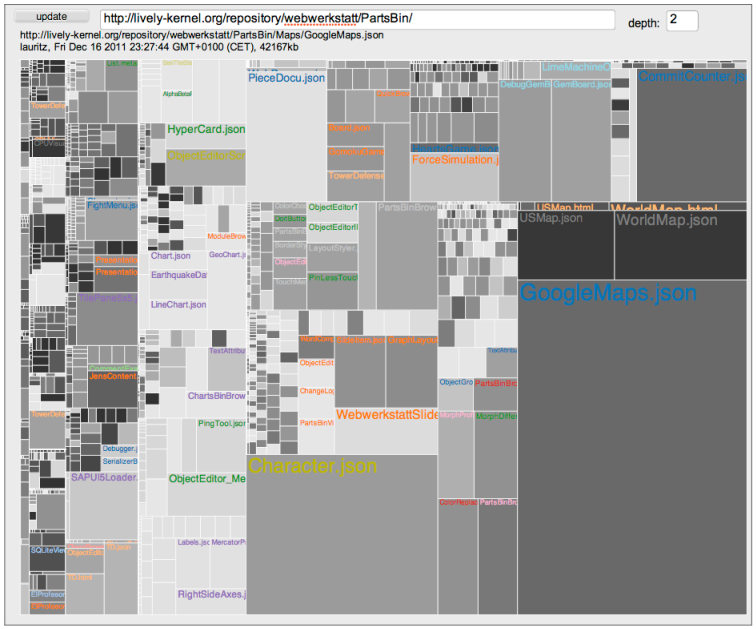
Lively Object Converter

- Migrate older Lively Kernel Objects / Worlds (JavaScript / DOM)
- Object structure and behavior
- Compatibility layers (Context-oriented Programming)



Lively4 Semantic Code Overview

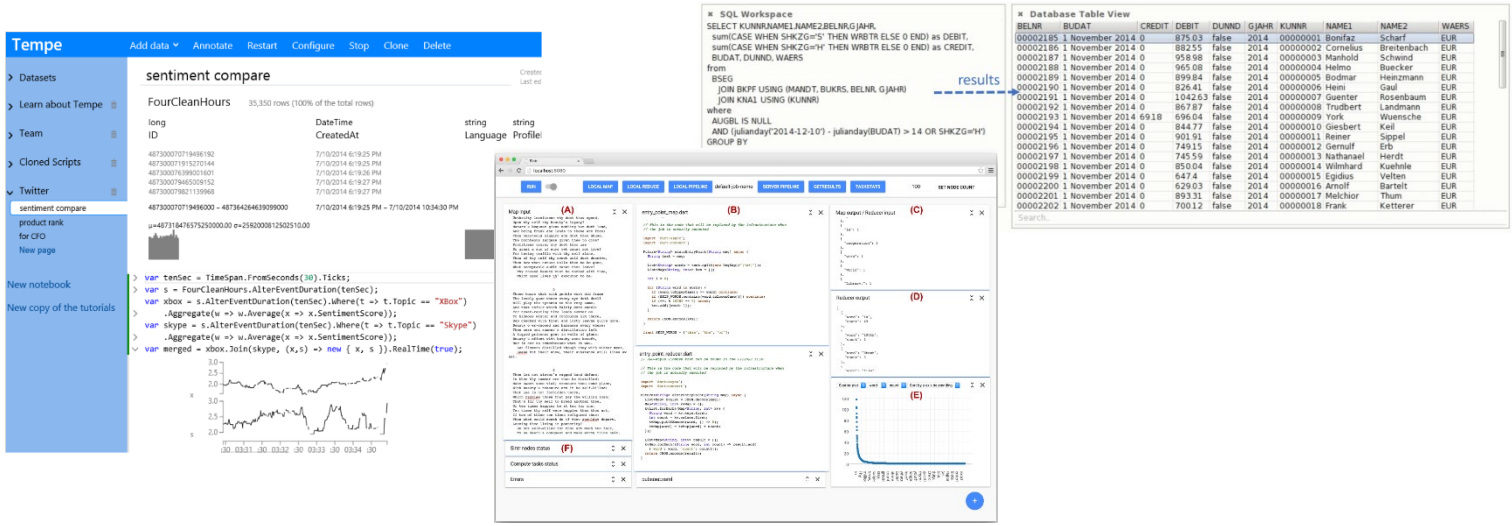
- Files vs. modules, classes, methods (JavaScript)
- Static and dynamic structure and dependencies
- Code coverage / system heat map



Exploring Even Large Databases

BSEG join BKPF join KNA1

KUNNR, MANDT, BUKRS, BELNR, GJAHR, WRBTR, SHKZG



- Working with legacy systems often involves understanding obscure data schemata
- Working in an with Smalltalk-style liveness eases the exploration process
- Large datasets would degrade the liveness

- Making use of streaming and sampling to enable a short feedback loop when working with lage data sets
- Algorithms and tools for characterizing the data and pointing out relevant aspects

“The Program according to Garp”

```
Diary of aCompiler(10293192)
-> aClassBrowser #compile: aText(201241923)
   <- aParser(12391029) #parse: aText(201241923)
      -> aParser(12391029) #instanceVariables
   <- anASTNode(95812092) #checkClassNames
   <- MethodCue #new
   <- aMethodCue(82934823) #ast: anASTNode(95812092)
```

- Who called me? Whom did I call? What did we exchange? In which context did we do that? What is my general role in the application?
- Tracing from the perspective of a single object
- Making the data accessible through a graphical tool allowing for browsing and querying

- Related work
  - Lienhard, Adrian, Stéphane Ducasse, and Tudor Gîrba. "Object flow analysis: taking an object-centric view on dynamic analysis." *Proceedings of the 2007 international conference on Dynamic languages: in conjunction with the 15th International Smalltalk Joint Conference 2007*. ACM, 2007.
  - Vasily Kirilichev, Eric Seckler, Benjamin Siegmund, Michael Perscheid, and Robert Hirschfeld.*Stepwise Back-in-time Debugging.* In Proceedings of GI Informatiktage 2014, Potsdam, Germany, March 27-28, 2014, GI.

Categorizing Message Categories

```
def set_color_space(self, color_space_name)
   self.color_space_name = color_space_name

# Predicates

def is_srgb(self):
   return self.color_space_name == 'srgb'
```

```
((SystemNavigation default allImplementorsOf: #expectedFailures)
                collect: [:m | m category]) asSet

"Prints" #(accessing characterization failures
                running setup testing 'testing-outer' tests)
```

- Code sections or message categories are useful for navigating large classes
- What could be useful metrics for finding relevant/interesting/problematic message categories?
  - Size
  - Heterogeneity (in how many other categories are message from this category?)
  - …
- How would an automatic refactoring tool for categories look like?
- Outlook: What if we had tags instead of categories which allowed us to browse the system from different angles?

Finding and Using Implied Protocols

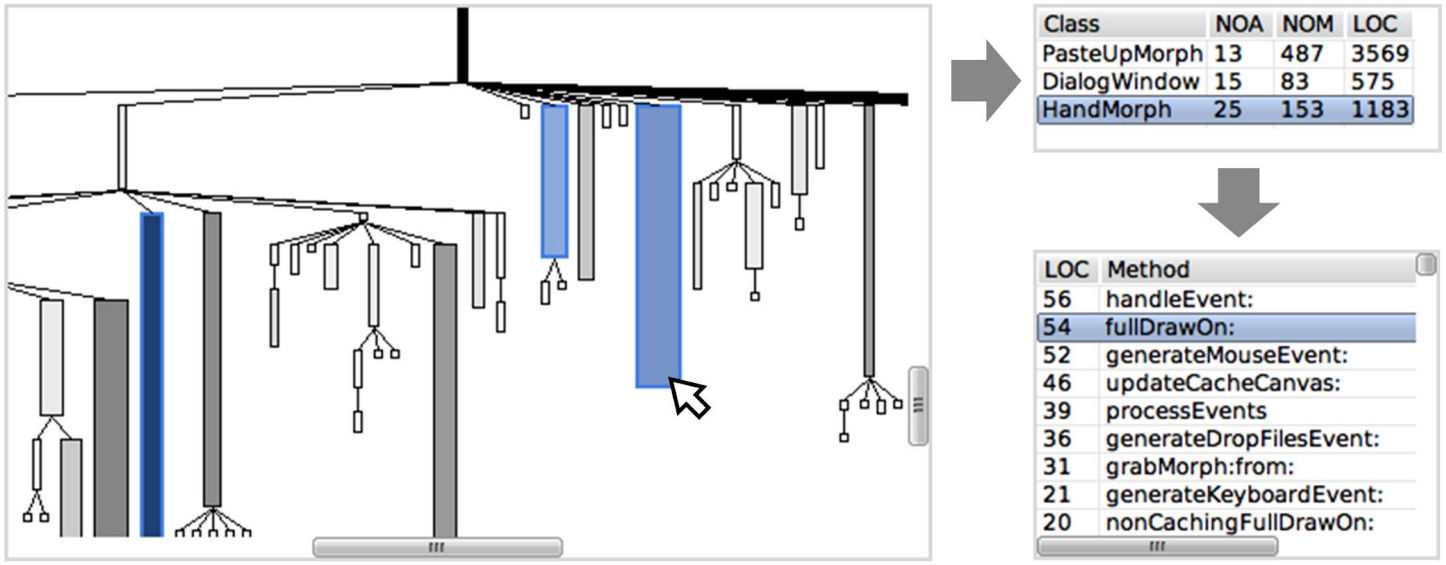
```
max: aMagnitude
   ^ self > aMagnitude ifTrue: [self] ifFalse: [aMagnitude]

> aMagnitude
   ^ aMagnitude < self

between: min and: max
   min <= self ifFalse: [ ^ false ].
   ^ self <= max

<= aMagnitude
   ^ (self > aMagnitude) not
...
```

- Some protocols rely on a very small set of primitive messages
- In single-inheritance systems they can be difficult to
  - Recognize (#between:and: only requires #< and #=
  - Implement (a class supports #do: but not #collect:)

- How to automatically determine semantically coherent implied protocols?
- How to provide guidance in implementing an implied protocol in a domain class?

Polymetric Views

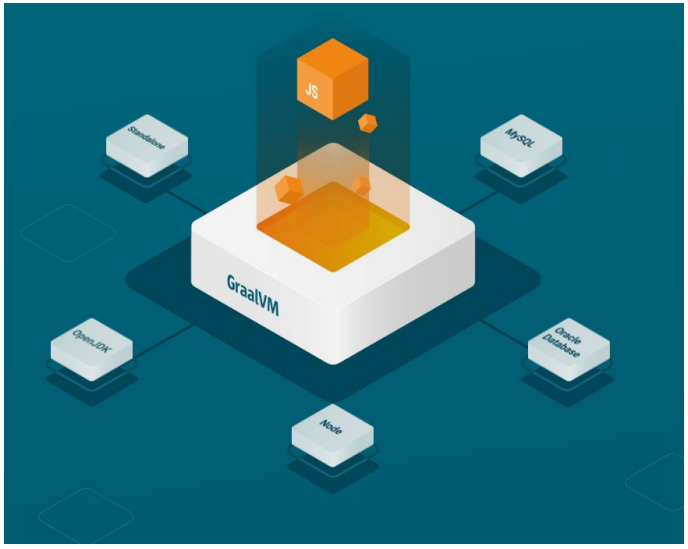
- Polymetric exploration tool in Squeak Smalltalk
- Software Visualization
- Packages, classes, methods + software metrics
- Integration with Smalltalk tools

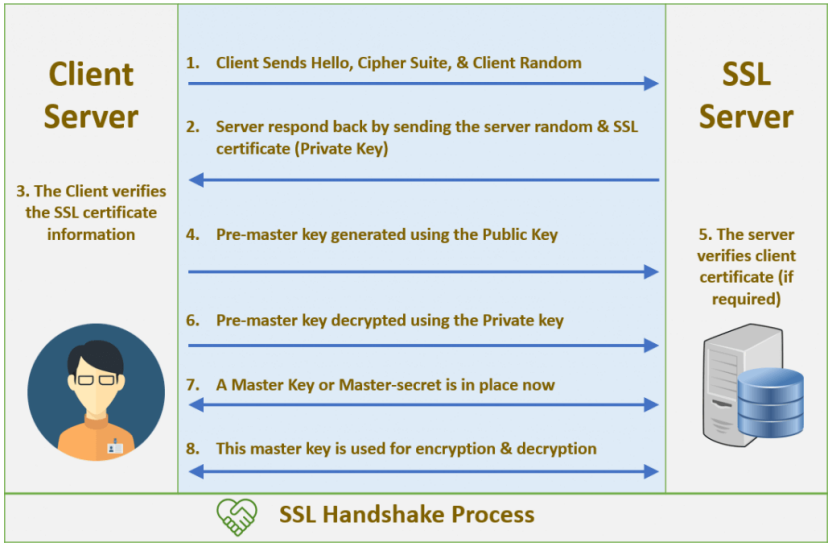


SSL Support in GraalSqueak

A Squeak/Smalltalk implementation for the GraalVM.

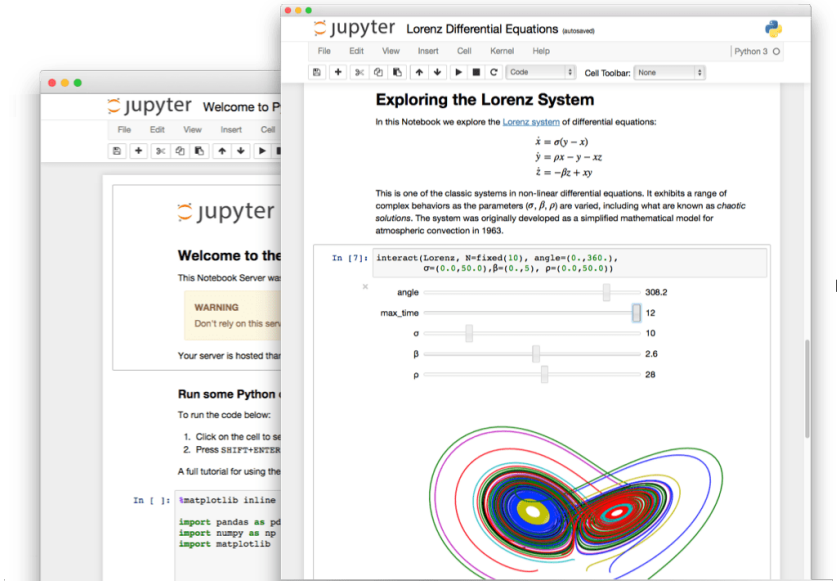
- Reverse-engineer SSL implementation in Smalltalk
  - Build tools for visualizing/debugging SSL handshake
  - **Goal:** Implement SSL plugin in Java for GraalSqueak

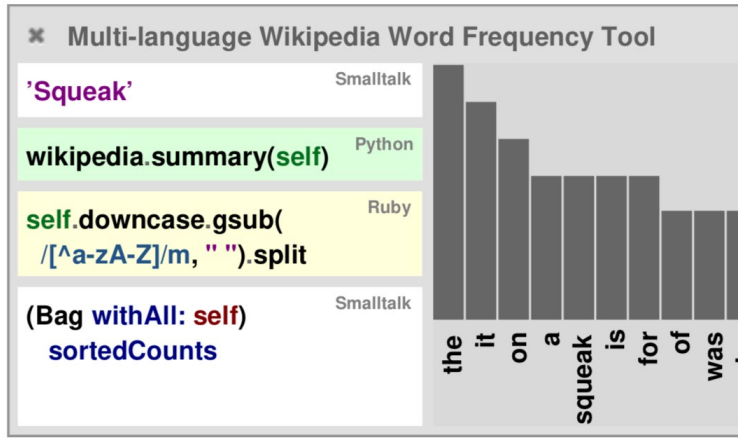




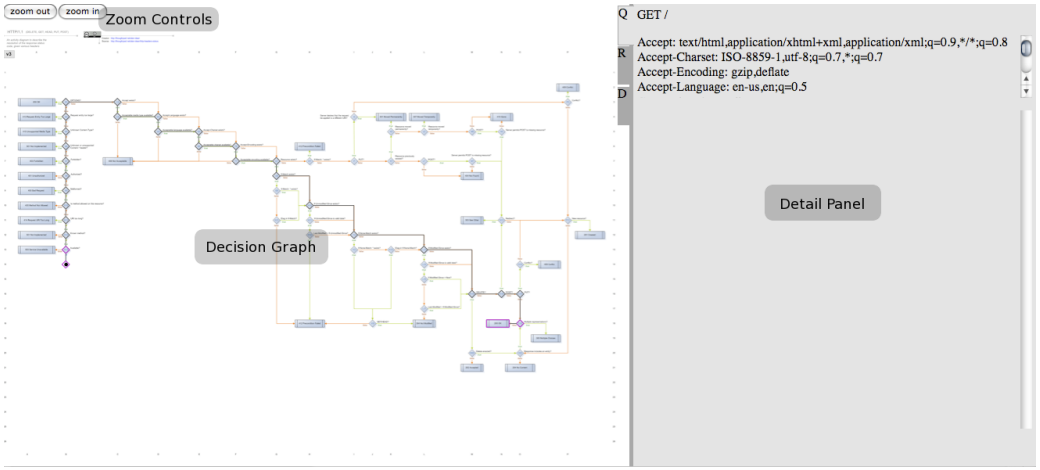
Jupyter Notebooks in Squeak/Smalltalk

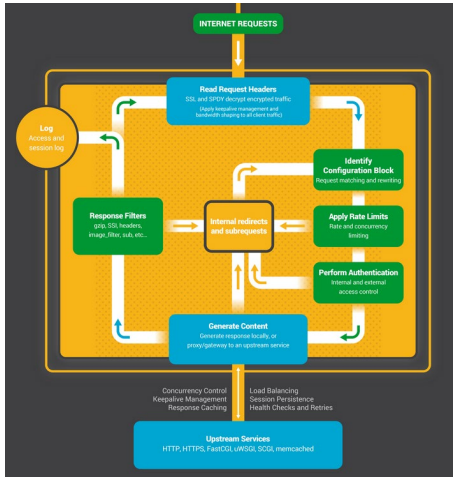
- Notebooks in Squeak/Smalltalk
  - Load and save (export) notebook files
  - Notebook UI, support for object inspector and charts, …
- Experiment with multi-language support in notebooks (Smalltalk, Ruby, Python, JavaScript, LLVM, …)





Exploring Web Request Processing



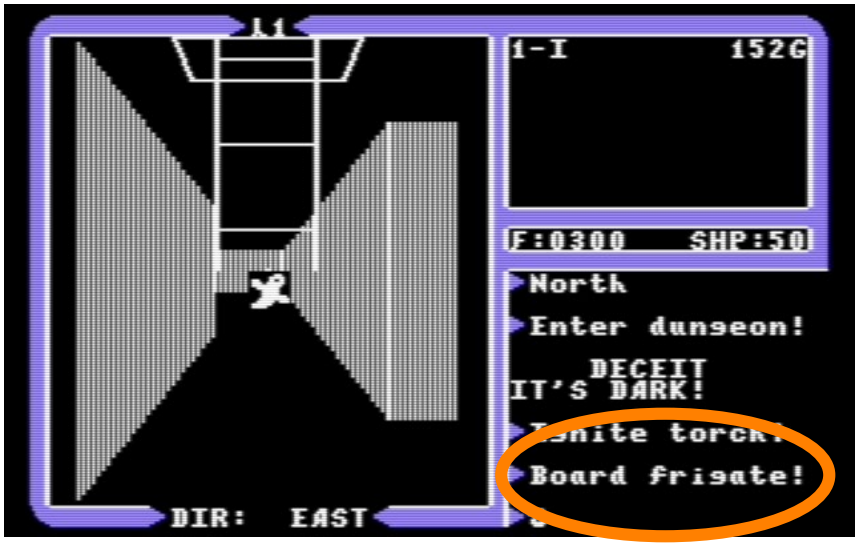


- Web server configurations define the processing steps a web server uses at run-time
- There are little tools to inspect the actual run-time steps and how they process the incoming requests
- Missing hot-loading results in a long feedback loop

- Potential servers: nginx, Apache
- Inspiration: Erlang webmachine

Fixing Old Software Bugs

- Old software is available as binary only
- How to explore the bugs and capabilities of the system? Can we infer navigation, content, system state information?



WAT?

From raw dissasembly ...

```
>C:4772  c2 ef e1 f2  e4 a0 bc ad  00

.C:477b  4C 89 41    JMP $4189
.C:477e  A5 48       LDA $48
.C:4780  C9 14       CMP #$14
.C:4782  F0 56       BEQ $47DA
.C:4784  C9 15       CMP #$15
.C:4786  F0 52       BEQ $47DA
.C:4788  20 21 08    JSR $0821

>C:478b  c2 ef e1 f2  e4 a0 00

.C:4792  A5 48       LDA $48
.C:4794  C9 10       CMP #$10
.C:4796  F0 16       BEQ $47AE
.C:4798  C9 11       CMP #$11
```

```
cmd_board:
        lda player_transport        ; This is $00, so we branch to @onfoot.
        cmp #$1f
        beq @onfoot
        jsr j_primm
        .byte "Board <-", $00

        jmp print_cant

@onfoot:
        lda tile_under_player       ; If the player is next to a ladder going up
        cmp #$14                    ; the current tile is $10.
        beq board_horse

                                    ; It's not a horse so the game prints "Board ".

                                    ; As it happens the ladder up dungeon tile is
                                    ; the same as a ship facing left on the
```

to annotatable, navigable code

https://magervalp.github.io/2015/04/02/u4-dungeon-frigate.html
https://gist.github.com/robey/1bb6a99cd19e95c81979b1828ad70612

Recovering Data

- New engines for old games
  - How do we recover the game data? What format is that in?
  - What does the binary tell us? Magic format bytes? Archives without headers? Run-length encoding? Endianness? Can we interactively explore, infer, and guide extraction?
- Example: Warcraft 1 DOS vs Mac version. Can we extract the dataset?
  - Drop the archive into a tool, start exploring subsets of the data, find the code, images, etc....

https://blogs.msdn.microsoft.com/oldnewthing/20180706-00/?p=99185
https://github.com/Wargus/war1gus/issues/78
http://www.blizzardarchive.com/pub/index.php?id=war1

Topics

- Xray through UI
- Scriptable Trace Viewer in Lively4
- Tracking JS Built-ins for Changes
- Static Analysis for High-Level Language Features
- Why and Why Not Questions
- Github Explorer
- Stroposcoping for non-graphical Objects
- Exploring Even Large Databases
- Lively Object Converter
- Lively4 Semantic Code Overview
- “The Program according to Garp”
- Categorizing Message Categories
- Finding and Using Implied Protocols
- Polymetric Views
- SSL Support in GraalSqueak
- Jupyter Notebooks in Squeak/Smalltalk
- Exploring Web Request Processing
- Fixing Old Software Bugs
- Recovering Data

Development Links

- Squeak
  - https://squeak.org/
  - https://github.com/hpi-swa/vivide
- Lively4:
  - https://github.com/livelykernel/lively4-core
  - https://lively-kernel.org/lively4/lively4-core/start.html

Hand-In Reference Sheet

- Presentation
  - Presentation (pdf)
  - Screencast (mp4, public)
- End of Semester (in zip, link or email)
  - Presentation and Screencast again
  - Abstract (txt)
  - Figure / Screenshot (png, 800x600)
  - Sourcecode (MIT License)
  - Data
  - (squeak image)
  - README.md
- All hand-ins will be archived in a public repository

Organization

- Course
  - Project-Seminar, 4 SWS, 2 students per group
- Presentations
  - Mid-term Presentation (Nov)
  - End-term Presentation (Jan, Feb)
- Grading
  - 6 ECTS graded credit points
  - Grade based on project work and presentation
- Important dates
  - Project topics on October 16th
  - Enrollment with preferred topic names on or before October 21th
    - Mail to stefan.lehmann@hpi.de and jens.lincke@hpi.de with RE18 in subject
  - Topic assignment on October 23th
  - Presentation dates determined after topics are assigned