Hierarchical Abstraction of Execution Traces for Program Comprehension

Kaj Dreef

Supervisors: prof. James A. Jones & prof. Arie van Deursen

Outline

1. Introduction
2. Motivation
3. Approach
4. Hierarchical Phase Visualization
5. Evaluation
6. Discussion
7. Conclusion
8. Demo - SageVis

Introduction

Introduction

- Software comprehension is essential, while performing maintenance tasks
- Understanding runtime behavior is complex, because execution traces:

    1. are long (i.e. 1.000.000's of events);
    2. contain many low level details;
    3. contain no abstractions.

Motivation

Motivation

- Challenge 1: Information overload
- Challenge 2: Behaviors Contain Sub-Behaviors
- Challenge 3: Incomprehensible Execution Traces

Approach

-- Sage --

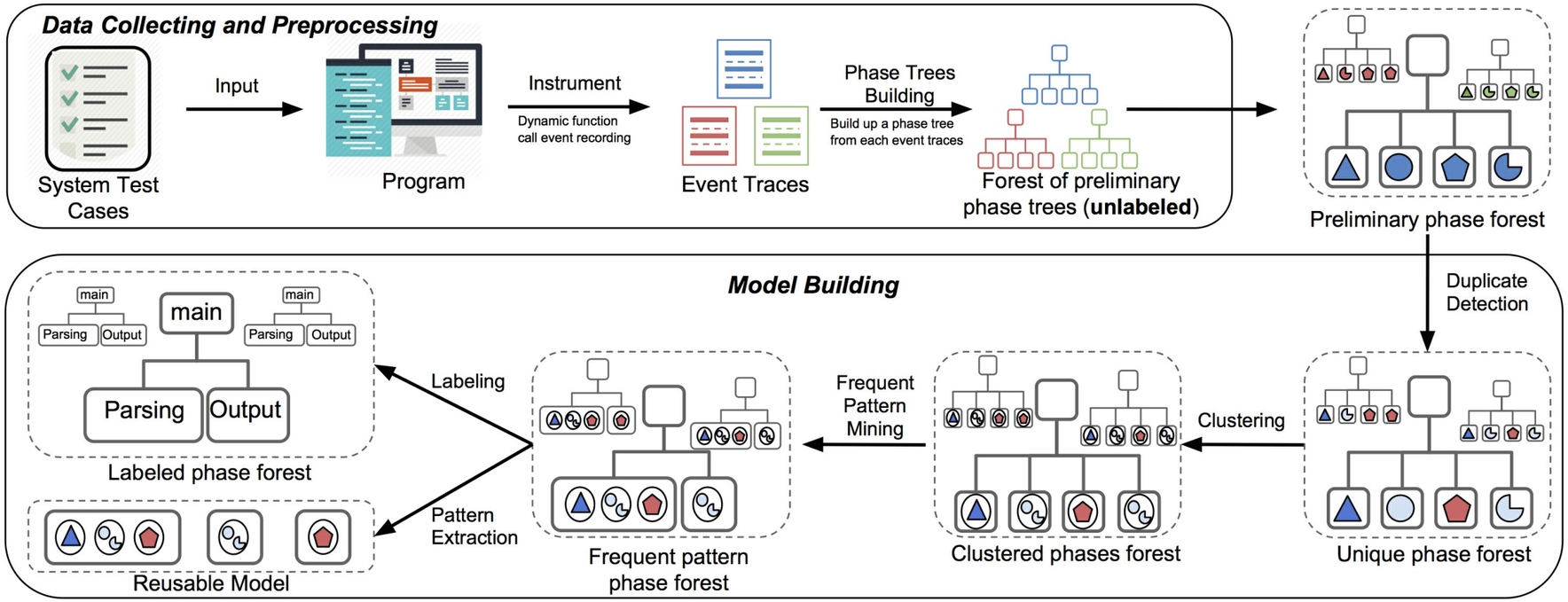
Sage

- Goal: Hierarchically abstracting an execution trace into high-level phases.

Sage

- Goal: Hierarchically abstracting an execution trace into high-level phases.
- Approach:

    1. Data Collection & Phase Detection
    2. Model Building & Execution Abstraction



Sage

Data Collection

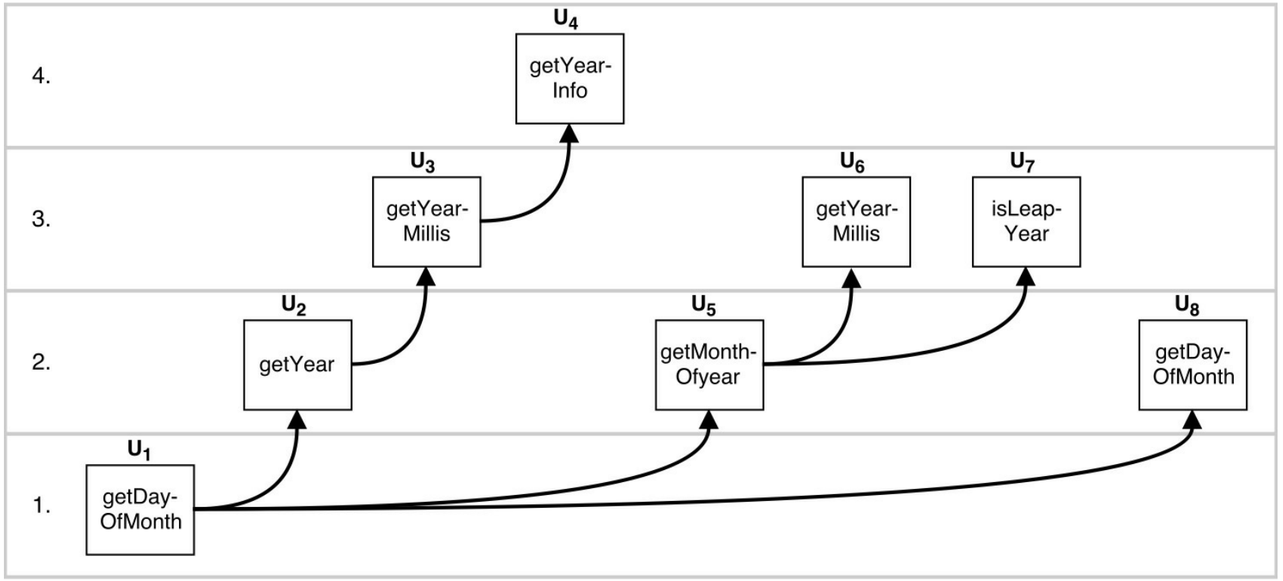
- Dynamic Instrumenter (Blinky)
- Collect only method-invocation enter events
- Each method-invocation event is captured as a triple of:
  - Execution ID
  - Method Signature
  - Call depth

Sage

Data Collection

Phase Detection

- Simplified steps of the phase detection:
  1. Input of phase detection is an execution trace

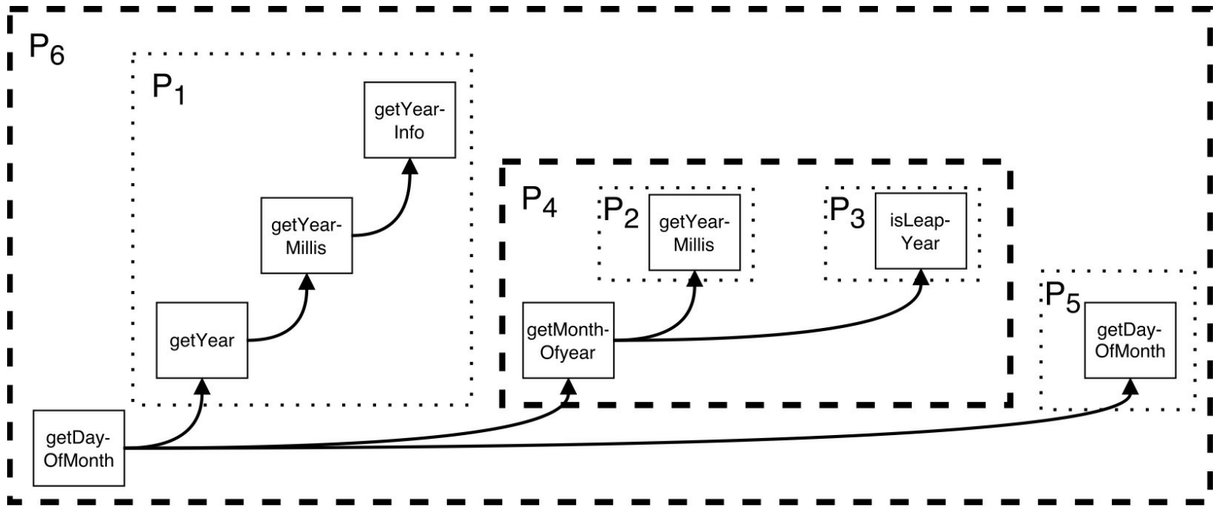


Sage

Data Collection

Phase Detection

- Simplified steps of the phase detection:
  1. Input of phase detection is an execution trace
  2. Use the call depth to locate the phase boundaries

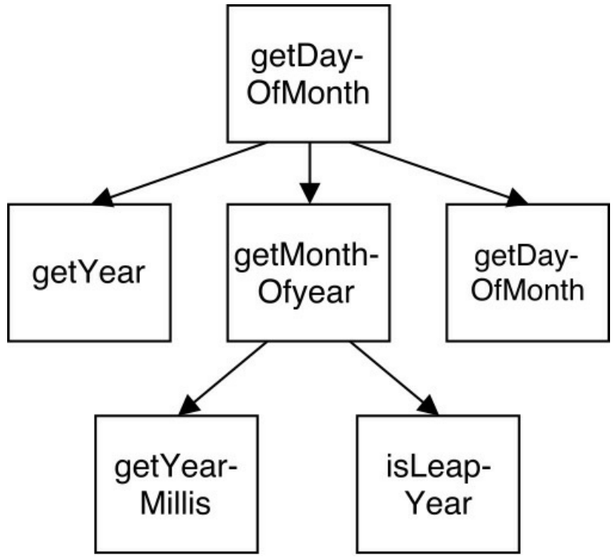


Sage

Data Collection

Phase Detection

- Simplified steps of the phase detection:
  1. Input of phase detection is an execution trace
  2. Use the call depth to locate the phase boundaries
  3. Finally, the result is a tree of phases



Sage

Data Collection

Phase Detection

- Algorithm

Algorithm: Main loop

Input : *eventList*: List of execution events
Output: *root*: The root node of the phase tree
Initialization: Initialize an empty stack *historyStack* to hold previous execution events or phases.
Initialize an empty *root* node.

```
1  begin
       /* End of file event signalizes the end of the execution trace.          */
2      eventList.append( end of file event with call depth of -1 )
3      foreach event u_i in eventList do
4          if historyStack.size() == 0 then
5              historyStack.push(currEvent)
6              continue
7          end
8          prevEvent ← historyStack.peek()
9          if prevEvent.getCallDepth() < u_i.getCallDepth() then
10             historyStack.push(u_i)
11         end
12         else if prevEvent.getCallDepth() == u_i.getCallDepth() then
13             historyStack.pop()
14             node ← createNode(prevEvent)
15             historyStack.push(node)
16             historyStack.push(u_i)
17         else if prevEvent.getCallDepth() > u_i.getCallDepth() then
18             root ← findPhasesInHistory(u_i, historyStack)
19         end
20     end
21     return root
22 end
```

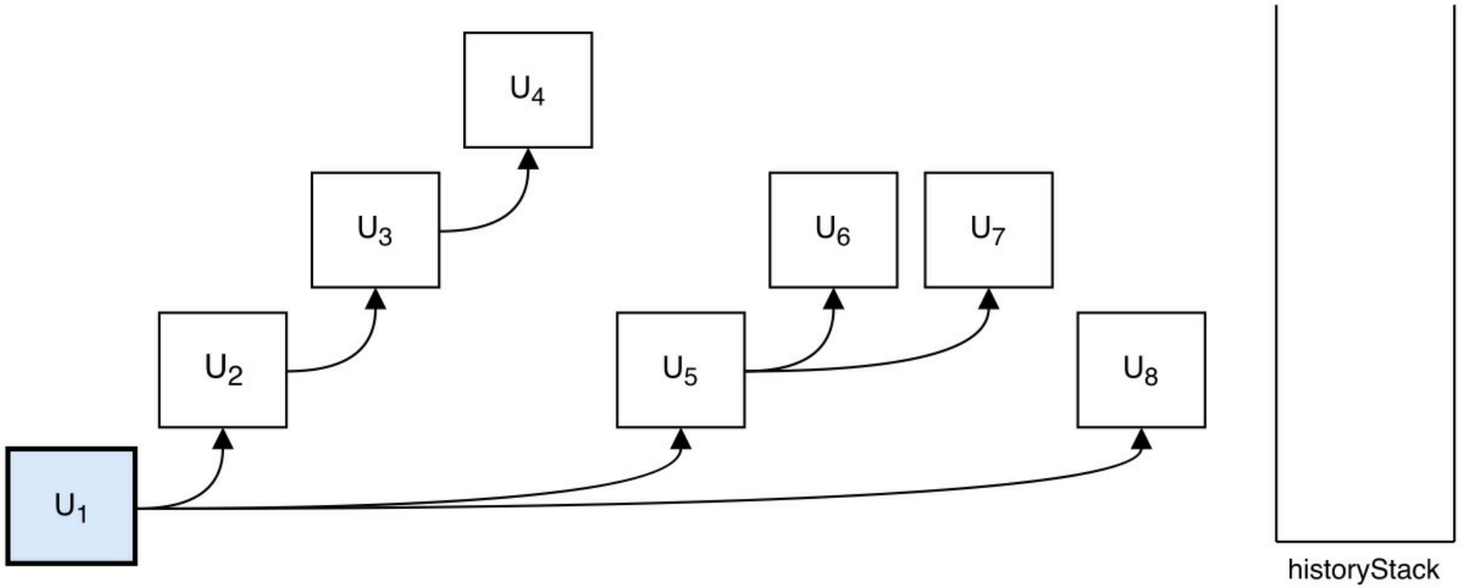
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 1: Initial trace



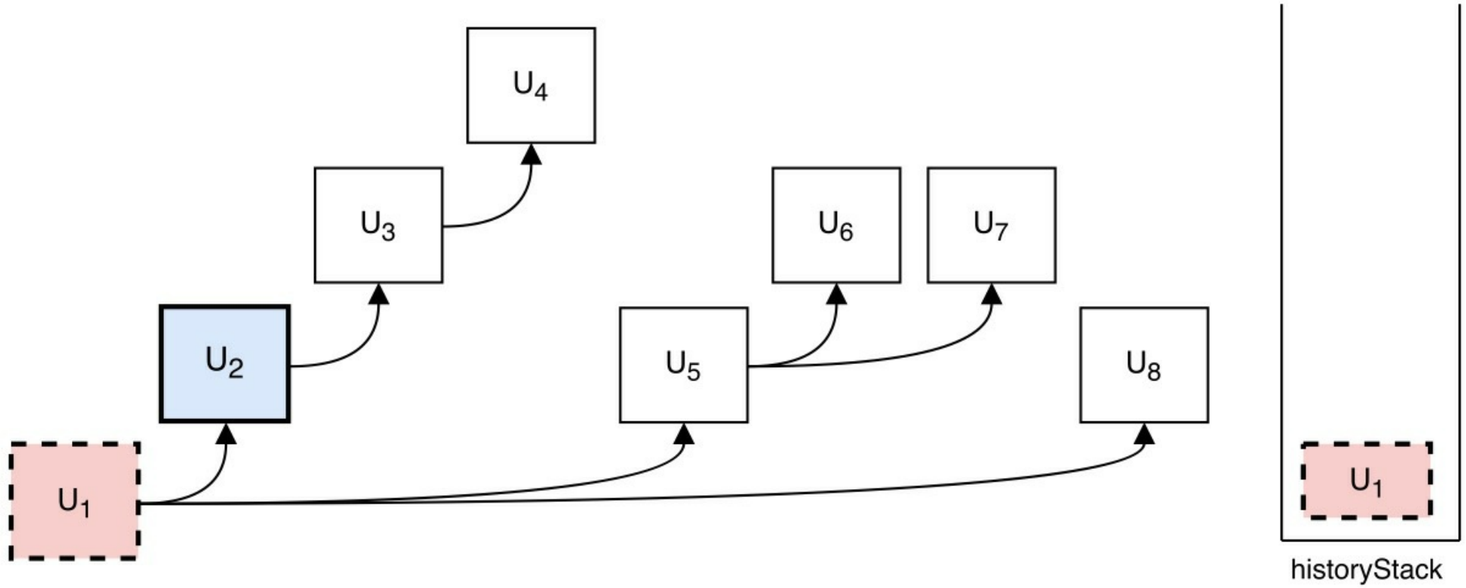
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 2:



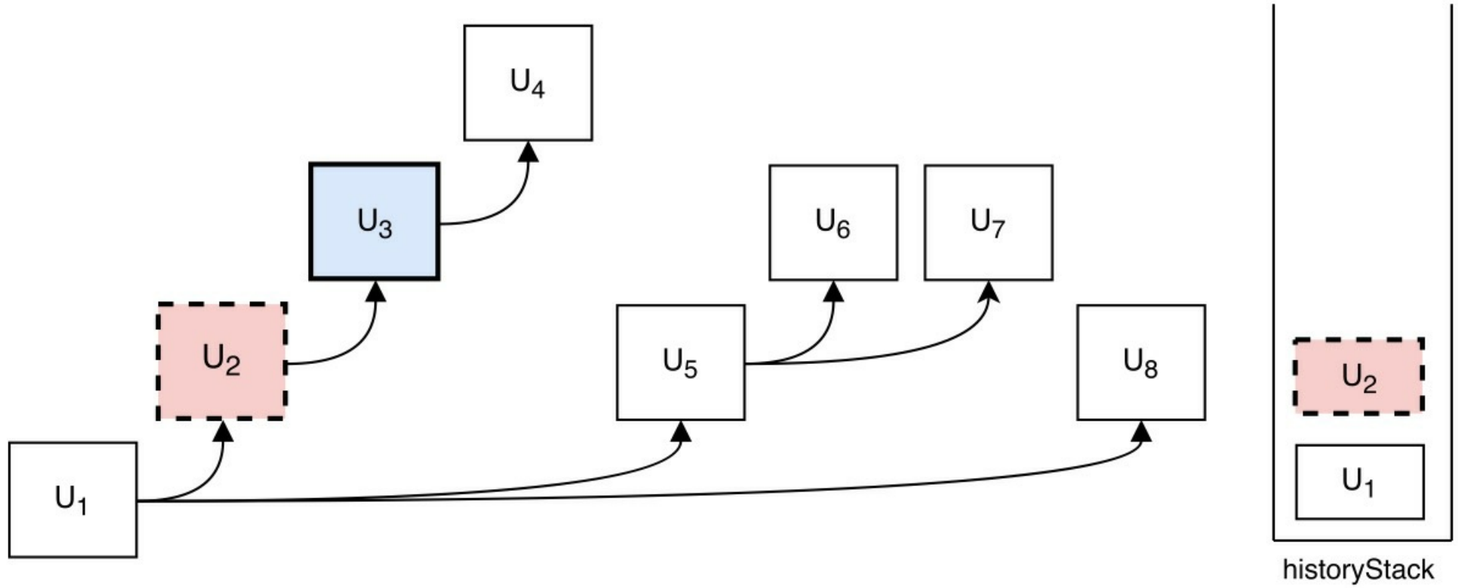
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 3:



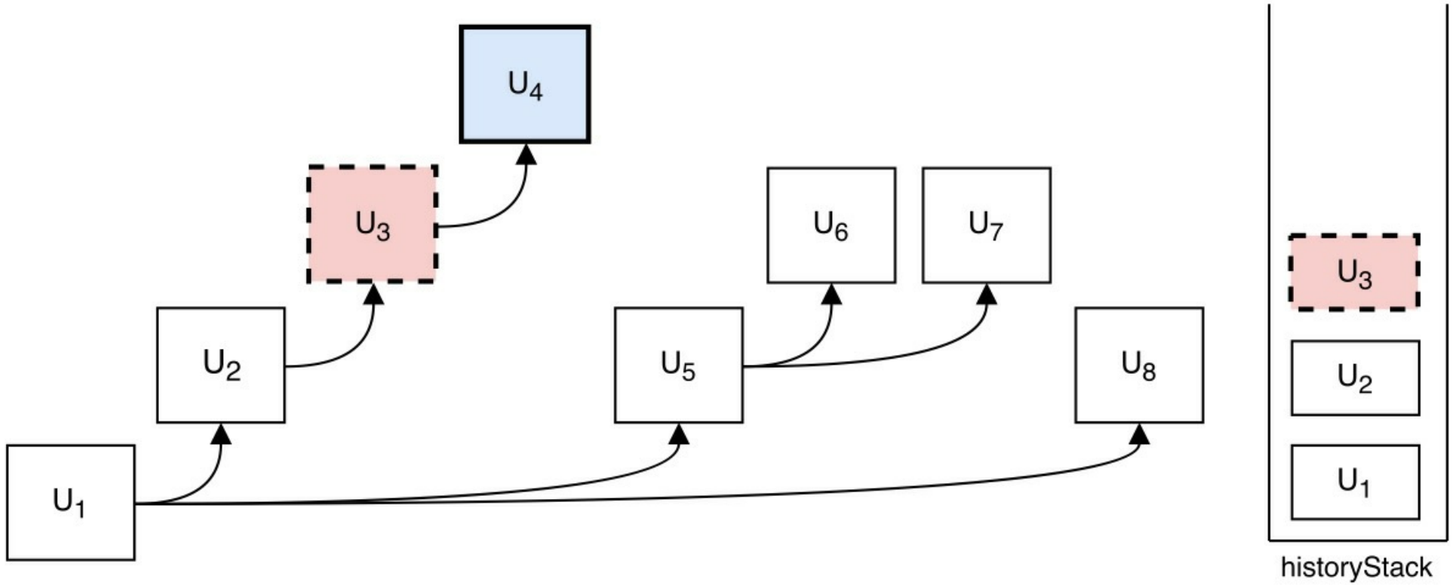
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 4:



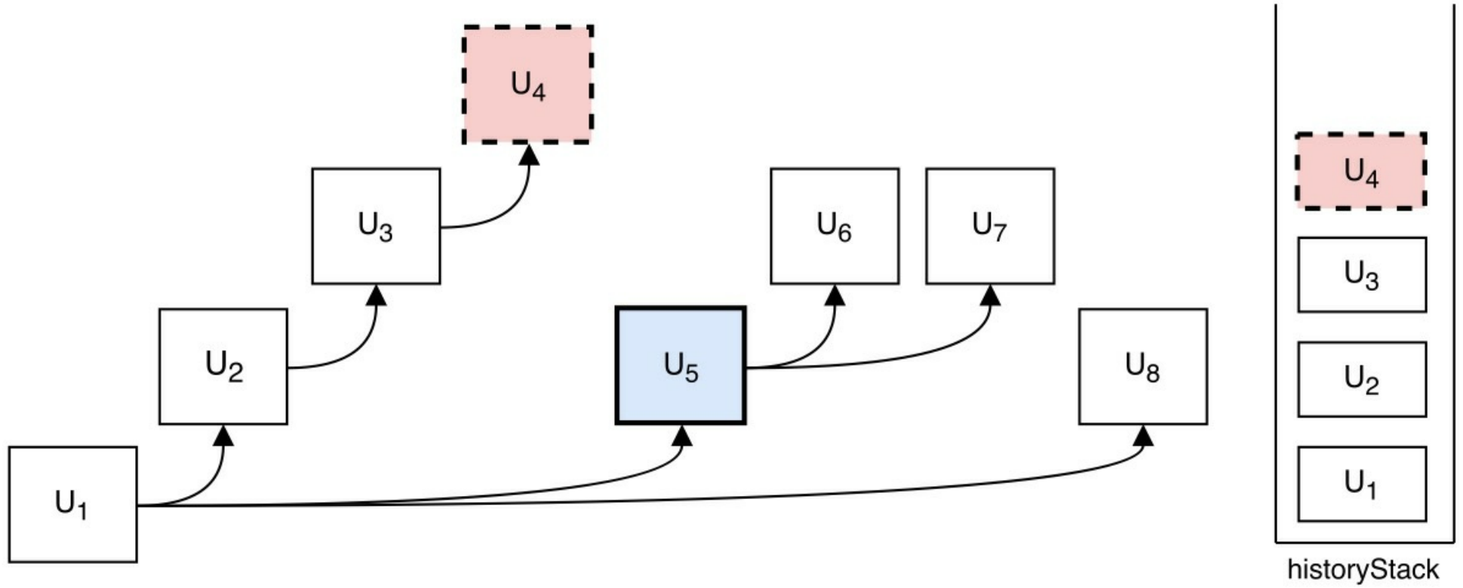
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 5: Stagnation!



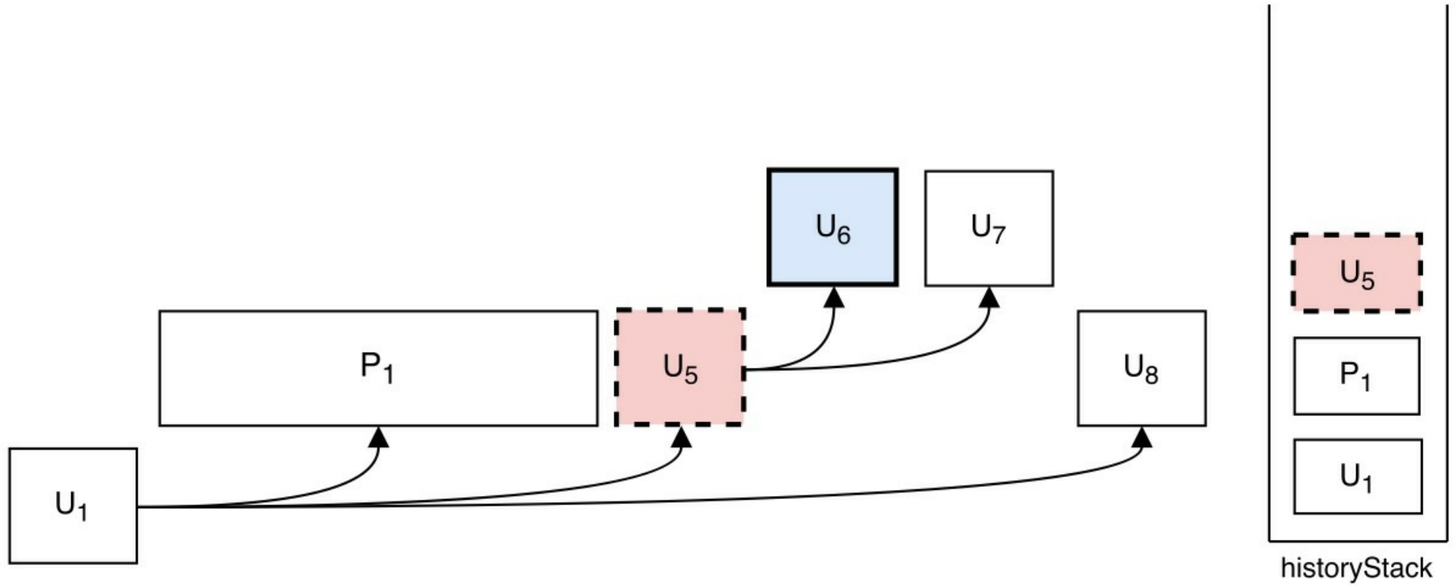
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 6:



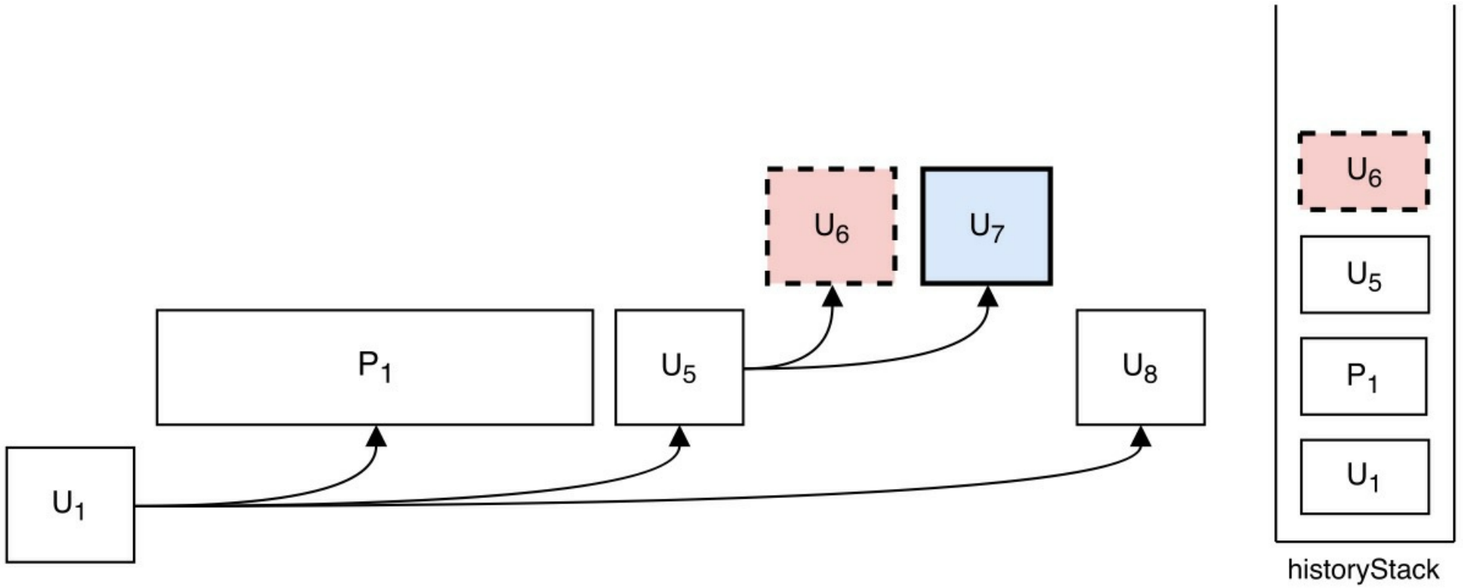
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 7: Stagnation!



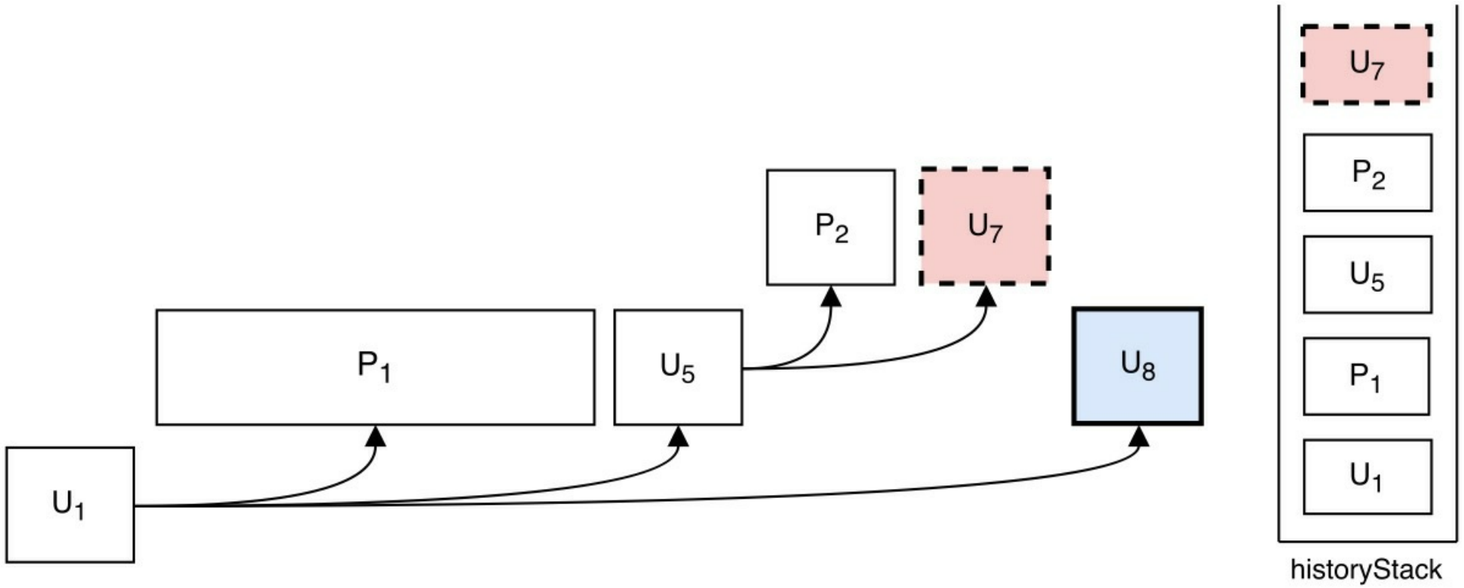
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 8: Stagnation!



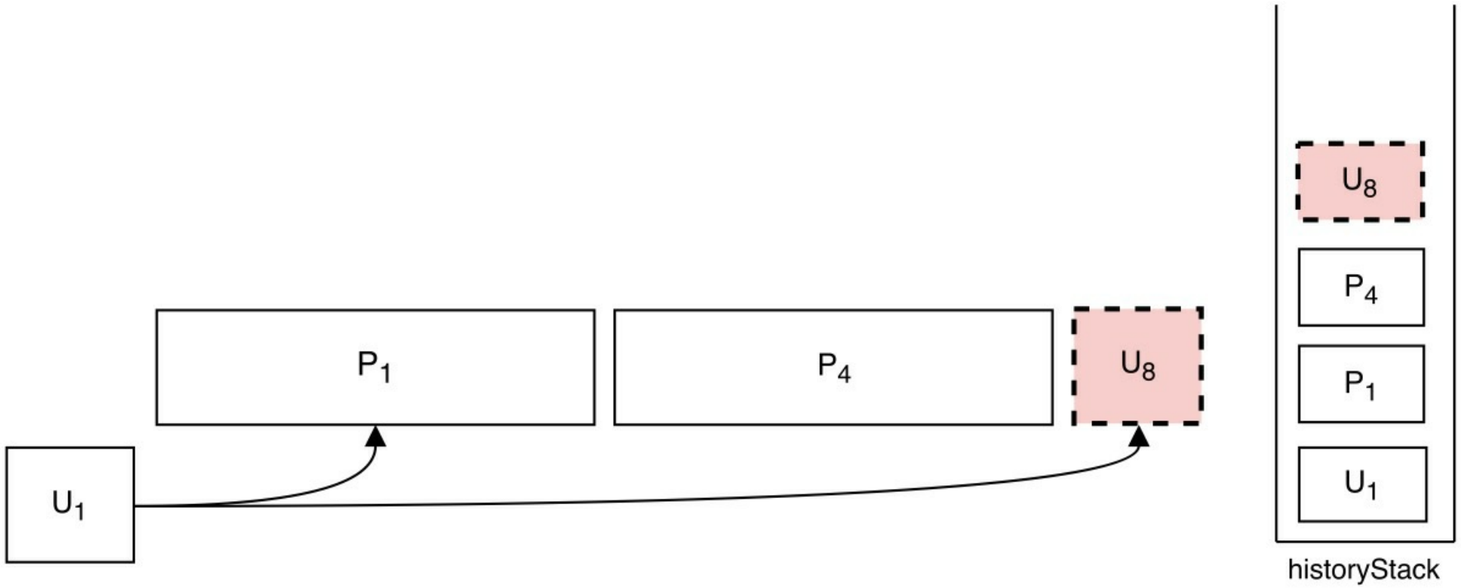
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Step 9: Stagnation!



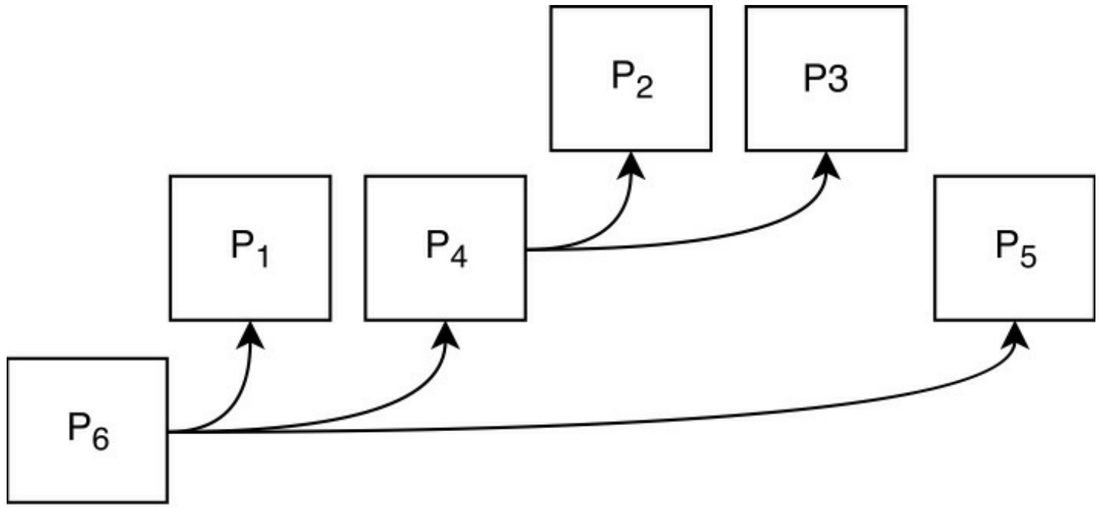
Sage

Data Collection

Phase Detection

- Algorithm
- Example

Finally: Preliminary Phase Tree



Sage

Data Collection

Phase Detection

Model Building

- Model is trained by a set of preliminary phase trees
- Consists of 4 steps:
  - Duplicate detection
  - Phase clustering (Agglomarative clustering)
  - Frequent pattern mining
  - Semantic labeling (using TF-IDF)
- The final model can be seen as a dictionary of phases

Sage

Data Collection

Phase Detection

Model Building

Execution Abstraction

- At this moment you have:
  1. a execution traces
  2. a model containing the recurrent phases
- Abstracting an execution trace needs steps:
  1. Phase detection
  2. Duplicate detection
  3. Apply the model
  4. If necessary, reapply labeling on phases that are not in the model.

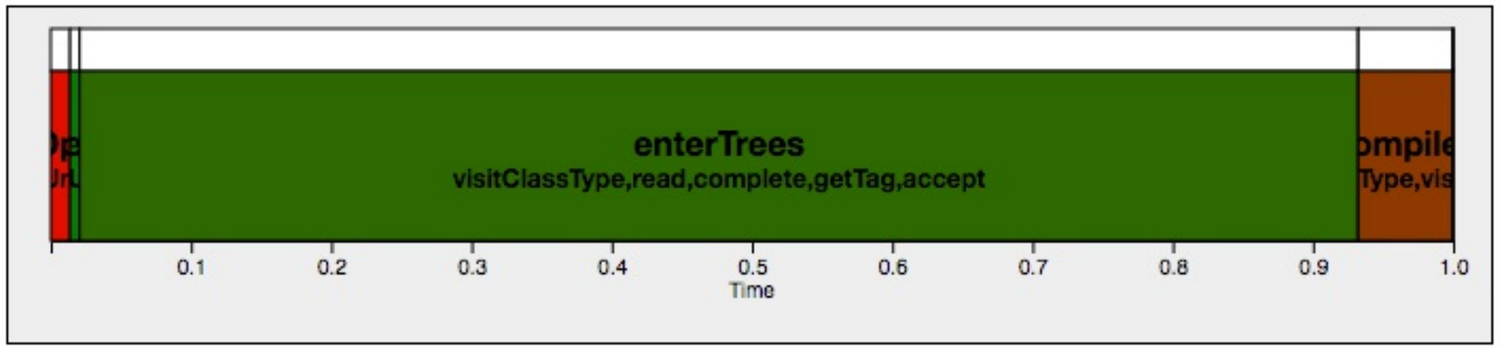
Hierarchical Phase Visualization

-- SageVis --

SageVis

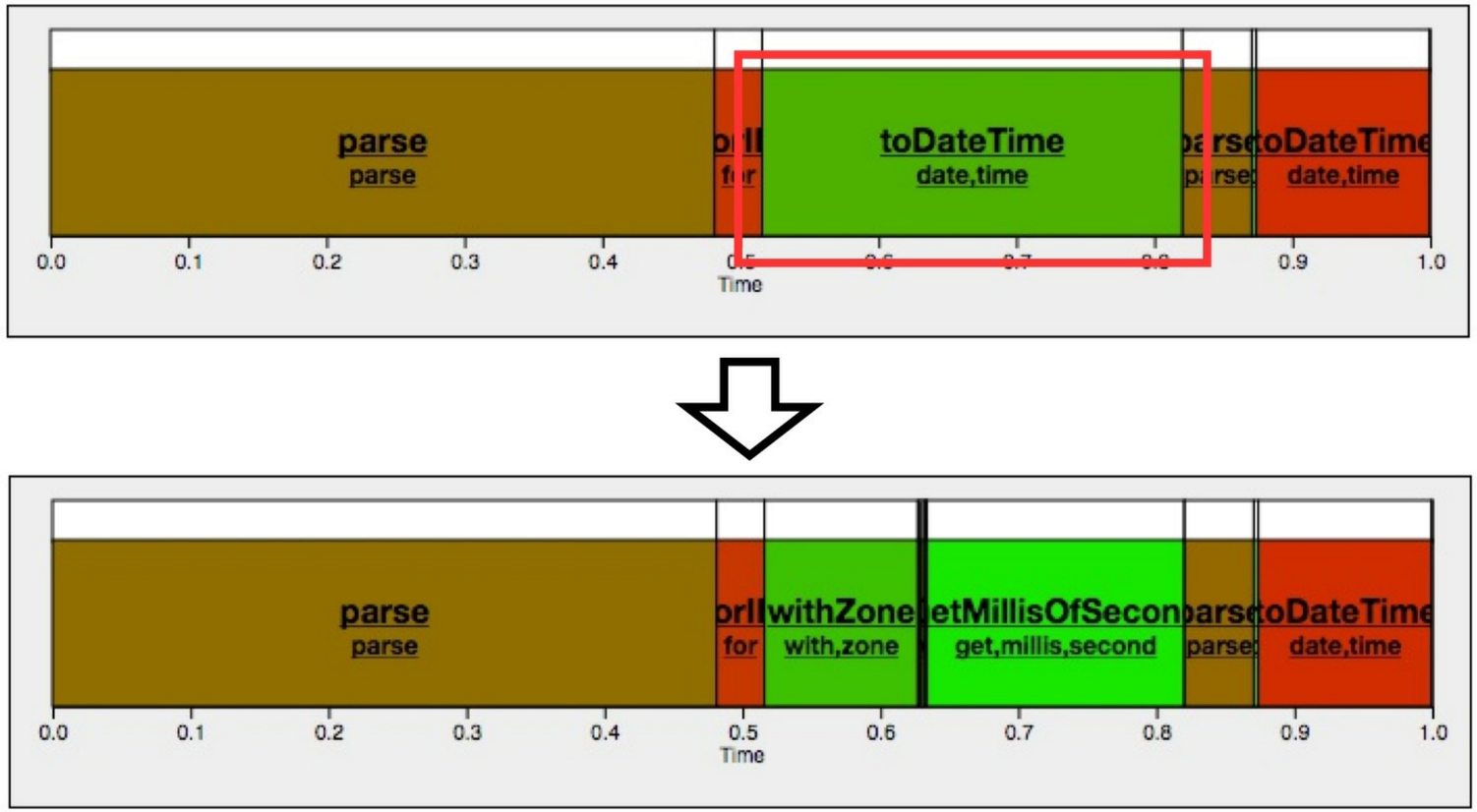
Overview

- Goal: Visualize the phases in an execution trace in an intuitive way, so developers can explore them at different levels of granularity.
- Overview of the visualization:
  - Horizontal axis presents time.
  - Width of a phase represents how long the phase took.
  - Vertical axis could display multiple threads.
  - Depth: behind every timeline another timeline can be displayed with a lower level of granularity
  - Color are used to more easily recognize similar phases.



[See www.github.com/kajdreef/sagevis](www.github.com/kajdreef/sagevis)

Expanding and Hiding Phases (1)



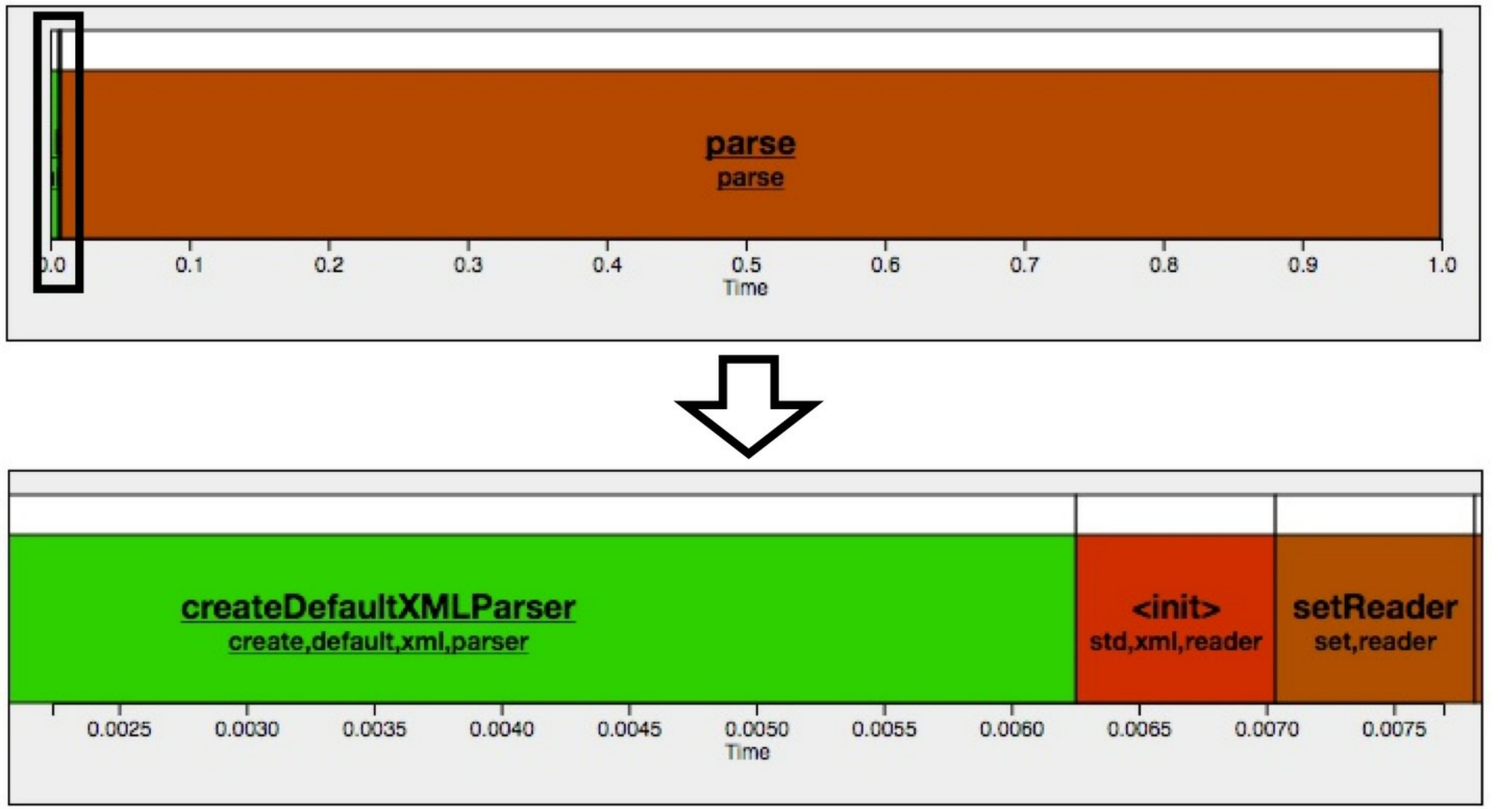
Expanding and Hiding Phases (2)

SageVis

Overview

Interactions

Zooming

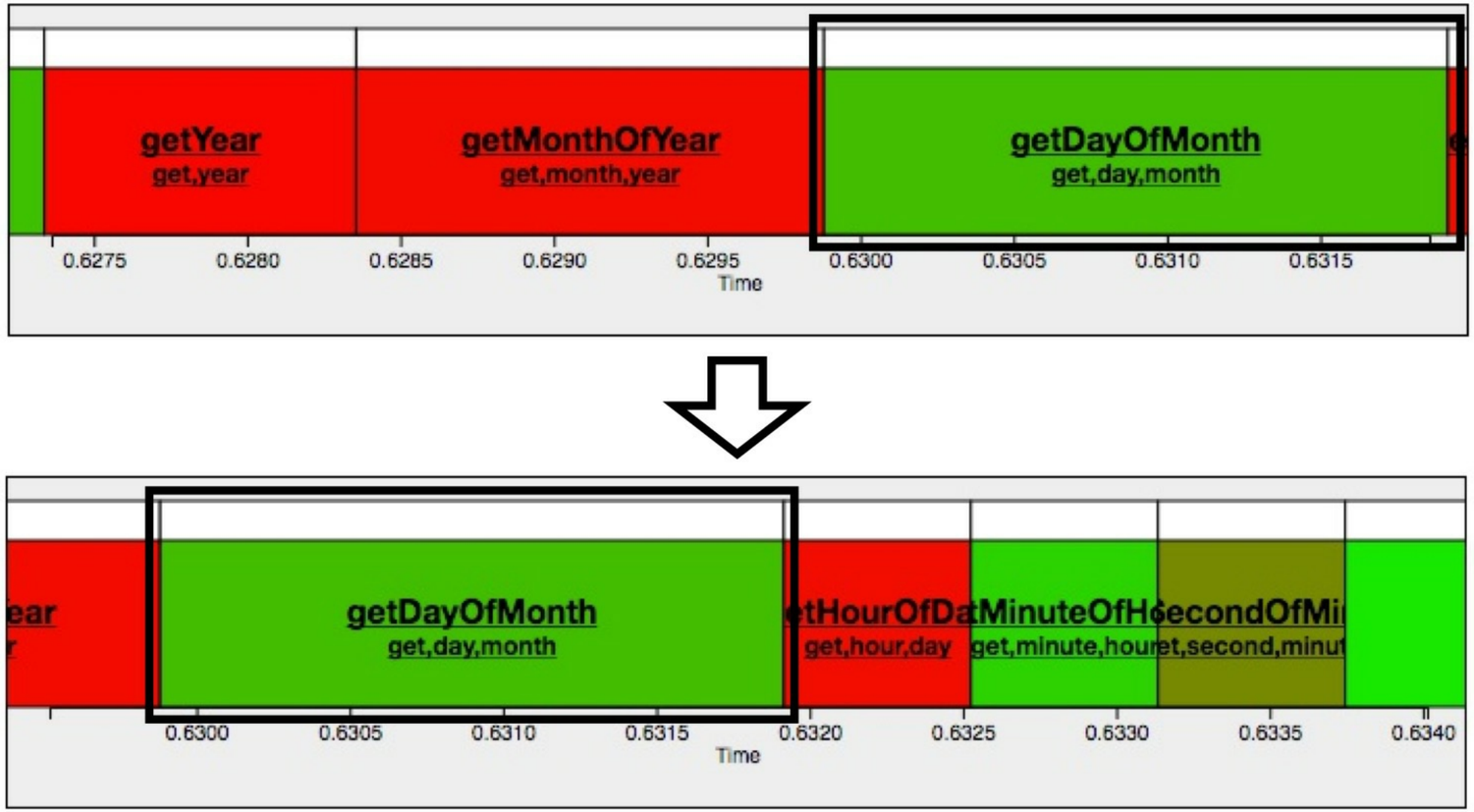


SageVis

Overview

Interactions

Panning

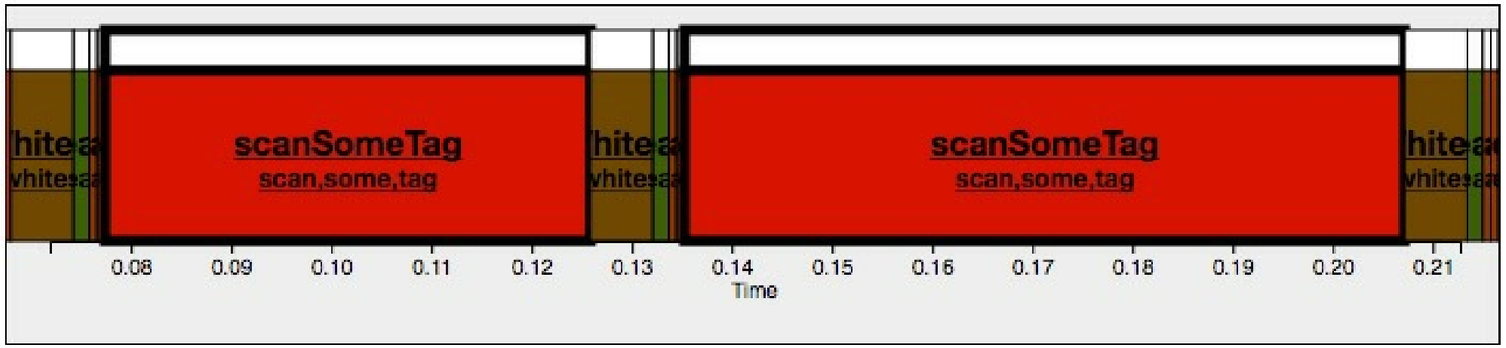


SageVis

Overview

Interactions

Phase Highlighting



Evaluation

Evaluation

Quantitative

RQ1: To what extent does SAGE alleviate the information overload challenge?

RQ2: Does the hierarchical phase abstraction provide substantially different levels of granularities of behavior?

Evaluation

Quantitative

RQ1: To what extent does SAGE alleviate the information overload challenge?

Projects	|Traces|	|Events|	|Methods|	Phase Detection |phases|	Duplicate Detection |phases|	Clustering |phases|	Pattern Mining |phases|
NANOXML	235	247,471	100	225,430	786	160	47
JAVAC	19	22,439,965	3919	51,878	1605	650	57
JEDIT	18	10,129,771	5431	692,286	17,088	3838	84

NanoXML phase distribution:

Level	Average	Median	Max	Min	Std	Skewness	Kurtosis
1	3.5	3.0	20	1	2.6	4.9	27.6
2	5.6	5.0	32	1	4.6	3.4	16.2
3	12.0	6.0	38	1	10.8	0.7	−1.0
4	12.5	6.0	42	1	11.7	0.8	−0.8
5	19.4	15.0	90	2	15.6	1.4	2.7
6	23.8	19.0	140	2	19.1	2.6	13.0
7	30.3	24.0	269	2	32.5	5.1	34.3
8	43.5	41.5	414	3	50.4	4.6	28.7

Evaluation

Quantitative

RQ2: Does the hierarchical phase abstraction provide substantially different levels of granularities of behavior?

NanoXML phase distribution:

Level	Average	Median	Max	Min	Std	Skewness	Kurtosis
1	3.5	3.0	20	1	2.6	4.9	27.6
2	5.6	5.0	32	1	4.6	3.4	16.2
3	12.0	6.0	38	1	10.8	0.7	−1.0
4	12.5	6.0	42	1	11.7	0.8	−0.8
5	19.4	15.0	90	2	15.6	1.4	2.7
6	23.8	19.0	140	2	19.1	2.6	13.0
7	30.3	24.0	269	2	32.5	5.1	34.3
8	43.5	41.5	414	3	50.4	4.6	28.7

Evaluation

Quantitative

User Study

RQ3: Are the derived labels sufficiently meaningful?

Evaluation

Quantitative

User Study

RQ3: Are the derived labels sufficiently meaningful?

Functionality	$\|Handouts\|$	$\|Correct\|$	Accuracy
New File	56	46	0.82
Change Font	28	21	0.75
Word Count	56	28	0.50
Close&Exit	70	56	0.80
Keyword Highlight	28	11	0.39
Open File	28	22	0.79
Typing	70	52	0.74
Search	56	36	0.64
Indent	28	24	0.86
Total	420	296	0.70

Discussion

Discussion

1. Addressing Information Overload

- Execution traces contain millions of events
- Sage output in the tens of phases

Discussion

1. Addressing Information Overload

- Execution traces contain millions of events
- Sage output in the tens of phases

2. Addressing Behavior Subsumption

- Sage output provides multiple levels of granularity

Discussion

1. Addressing Information Overload

- Execution traces contain millions of events
- Sage output in the tens of phases

2. Addressing Behavior Subsumption

- Sage output provides multiple levels of granularity

3. Addressing Comprehensibility of Execution Traces

- Labeled phases enabled developers to comprehend the behaviors of the execution
- Average accuracy of 70%

Conclusion

Conclusion

- We introduced:
  - Sage: a technique that creates a hierarchical abstraction and multiple levels of granularity of an execution trace.
  - SageVis: an interactive visualization that enables developers to explore the execution trace at different levels of granularity.
- The quantitative evaluation revealed us:
  1. Sage can successfully abstract the execution trace and reduce the amount of data to be inspected
  2. Training costs are not inconsequential, but can be incurred infrequently and offline
- The user study showed us that for most functionalities the users were able to achieve a 70% accuracy.
- The case study demonstrates Sage ability to reveal the primary behavior phases within a large execution

Demo of SageVis

SageVis - javac trace

Extra Slides!

Visualization

SageVis

Data Structure

- Frequent pattern mining breaks the tree structure
  - Without FPM limits the reduction in phases
  - In the future explore boundaries?

Future work

Future work

- Overall

- Multithreaded support
- Evaluate Sage and SageVis on larger systems

- Visualization

- Dynamically Inferred State Machine

Evaluation

Quantitative

User Study

Visualization Comparison

RQ4: What is the impact of frequent pattern mining on the visualization?