"It’s hard to debug failures in the field"

Distributed across time and space

Hardware and software variation

Users are not software testers
Users can’t report failures accurately

end-user encounters bug in production code

files bug report with ad-hoc information

“a piece doesn’t rotate properly!!”

developer unable to reproduce the bug

Works for me..
Users can’t report failures accurately.

“The most severe problems were errors in steps to reproduce and incomplete information.”

Users can’t report failures accurately

Bug reporters and developers want better tool support for reproducing buggy behavior.

“What makes a good bug report”. Zimmerman et al. TSE Vol. 36, No. 5, 2010
Existing tools are imprecise and hard to use

macro replay
(CoScripter, Selenium, Sikuli)

Capture and simulate user input. Designed for test and task automation.

CoScripter
Leshed et al, CHI 2008

Selenium/WebDriver

Sikuli Script
Yeh et al, UIST 2009
Existing tools are imprecise and hard to use

**macro replay**
(CoScripter, Selenium, Sikuli)

Capture and simulate user input. Designed for test and task automation.

**deterministic replay**
(Mugshot, WaRR)

Save and reuse nondeterministic inputs to exactly recreate a specific execution.

Nondeterministic. Requires extra setup ahead of time. Can’t use with a debugger.

Play/pause buttons only. Slows down execution. Can’t use with a debugger.
Timelapse: a precise, fast, integrated replay tool

This talk:

- An interface for capturing and replaying program behavior
- Techniques for cheap, precise record/replay in web browsers
- How developers use record/replay during debugging tasks
How to capture program behavior
How to navigate a recording
Using replay while debugging
Browsers interpret input, render output
Timelapse captures a browser’s inputs

Browser Input
(User, Network, Timers)

Web Interpreter
(WebKit, Gecko)

Outputs Log

Input

Output
Timelapse replays a browser’s inputs

Web Interpreter (WebKit, Gecko)

Inputs Log

Output
Browsers have layered architectures

User input, commands

Policy decisions

Web Interpreter (WebKit, Gecko)

Date.now, win.colorDepth

Event loop callbacks

Embedders (Firefox, Safari, Chrome)

Platforms

15
Timelapse intercepts input at layer boundaries

Embedders
(Firefox, Safari, Chrome)

Web Interpreter
(WebKit, Gecko)

Platforms

Mac
GTK
Qt
Inspired by VM record/replay

VM record/replay

- Hardware interrupts
- Nondeterministic instructions
- Instruction counts

Browser record/replay

- Async callbacks
- Nondeterministic APIs
- DOM event counts
Memoizing nondeterministic APIs

During normal execution, `Date.now()` returns the current time.

```cpp
// file: Source/wtf/DateMath.h
inline double jsCurrentTime()
{
    return floor(WTF::currentTimeMS());
}
```
Memoizing nondeterministic APIs

During recording, the return value of `Date.now()` is saved.

```cpp
// file: Source/wtf/DateMath.h
inline double jsCurrentTime()
{
    return floor(WTF::currentTimeMS());
}
```
Memoizing nondeterministic APIs

On replay, the logged return value of `Date.now()` is used.

```cpp
inline double jsCurrentTime()
{
    return floor(WTF::currentTimeMS());
}
```
Making callbacks deterministic

while (true) {
    var event = queue.pop();
    this.dispatchEventToListeners(event);
}

Problem: accurately capturing and simulating event loop dispatches.
Making callbacks deterministic

Callback registered

enqueue()

while (true) {
    var event = queue.pop();
    this.dispatch(event);
}

timerFired()

timer 42,
34 DOM events

Callback executes

Inputs Log
Making callbacks deterministic

```
while (true) {
    var event = queue.pop();
    this.dispatchEventToListeners(event);
}
```

Inputs Log

Callback registered

34 DOM events!

Callback executes
Runtime overheads are acceptable

![Bar chart showing run times (multiple of baseline) for different applications. The x-axis represents various applications including JS Raytracer, Space Invaders, Mozilla.org, CodeMirror, Colorpicker, and DuckDuckGo. The y-axis represents the run times as multiples of the baseline. The chart includes bars for Baseline, Recording, 1× Replay, and Seeking.](image-url)
Recordings are small and compressible

<table>
<thead>
<tr>
<th>Site</th>
<th>recording duration (s)</th>
<th>resources on page (KB)</th>
<th>log growth (KB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSLinux</td>
<td>10.5</td>
<td>4500</td>
<td>0.8</td>
</tr>
<tr>
<td>JS Raytracer</td>
<td>6.3</td>
<td>5.9</td>
<td>1.6</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>25.8</td>
<td>712</td>
<td>2.2</td>
</tr>
<tr>
<td>Mozilla.org</td>
<td>22.3</td>
<td>2800</td>
<td>1.3</td>
</tr>
<tr>
<td>CodeMirror</td>
<td>16.6</td>
<td>168</td>
<td>1.0</td>
</tr>
<tr>
<td>Colorpicker</td>
<td>15.3</td>
<td>577</td>
<td>1.7</td>
</tr>
<tr>
<td>DuckDuckGo</td>
<td>14.1</td>
<td>1900</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Page resources dominate recording size

<table>
<thead>
<tr>
<th>Site</th>
<th>recording duration (s)</th>
<th>resources on page (KB)</th>
<th>log growth (KB/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSLinux</td>
<td>10.5</td>
<td>4500</td>
<td>0.8</td>
</tr>
<tr>
<td>JS Raytracer</td>
<td>6.3</td>
<td>5.9</td>
<td>1.6</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>25.8</td>
<td>712</td>
<td>2.2</td>
</tr>
<tr>
<td>Mozilla.org</td>
<td>22.3</td>
<td>2800</td>
<td>1.3</td>
</tr>
<tr>
<td>CodeMirror</td>
<td>16.6</td>
<td>168</td>
<td>1.0</td>
</tr>
<tr>
<td>Colorpicker</td>
<td>15.3</td>
<td>577</td>
<td>1.7</td>
</tr>
<tr>
<td>DuckDuckGo</td>
<td>14.1</td>
<td>1900</td>
<td>2.1</td>
</tr>
</tbody>
</table>
How would developers use it?

**Study Design**

- 20+ developers with industry experience
- within-subjects, 2 tasks per person, 45 minutes per task, 4 treatments

**Reproduction**

- RQ: changes to frequency/duration?

**Performance**

- RQ: complete tasks more quickly? more successfully?
- Who? Why?
How did developers use it?

**Study Design**

20+ developers with industry experience within-subjects, 2 tasks per person, 45 minutes per task, 4 treatments

**Reproduction**

Shorter and more frequent repro actions; Time spent unchanged (max. 25%; avg. 15%); Successful developers quickly integrated replay into their existing workflows.

**Performance**

Unsuccessful developers who used opportunistic strategies were distracted.
Current & Future Work

Visualizations

Interaction histories aid navigation, but not program understanding.

Passive capturing

Precision and low overhead don’t matter if you forget to start capturing.

Post-hoc analysis

Developers can gather more runtime data without reproducing behavior:

Post-hoc logging, Post-hoc Whyline, Post-hoc SeeSS, Testcase extraction
## Conclusion

<table>
<thead>
<tr>
<th>Record/Replay</th>
<th>Virtual machine replay techniques work well when applied to web applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualizations</td>
<td>Interaction histories supported—but didn’t reduce—reproduction of program state.</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Replay infrastructure enables new research, tools and workflows.</td>
</tr>
</tbody>
</table>

github.com/burg/timelapse
Replay fidelity and completeness

Divergence detection supports piecewise implementation.

Web interpreters expose a large and ever-changing API.

Timelapse doesn’t tame all sources of nondeterminism (yet).

Excepting untamed sources, the DOM tree and JavaScript heap are identical for all recorded and replayed executions.

Possible divergence is automatically detected when:

- DOM event counts differ on capture and replay
- Memoized inputs are overused or unused
- Network request details differ unexpectedly
- Known-bad APIs are used by client code
Interpreter inputs by source

**User**: mouse, keyboard, scroll, resize

**Network**: images, scripts, HTML, AJAX

**Commands**: page navigation

**Internal nondeterminism**: Animations, transitions, multimedia, async script and parser yields

**Functions**: Date.now, Math.random, etc

**Caching**: resources, cookies

**Timers**: timer schedule
Shim: the thing in the middle

Shims are used to implement deterministic record/replay.

The hard part of implementing record/replay is designing and placing shims.
Embedding and platform APIs

Abstraction layers separate web interpreters from platforms/embedders.
Embedding and platform APIs

Abstraction layers separate web interpreters from platforms/embedders.
Embedding and platform APIs

Shims sit between the web interpreter and abstraction layers.
Embedding and platform APIs

**Shims** sit between the web interpreter and abstraction layers.