Record, Replay, Rinse, & Repeat: Easily Rebuilding Programmatic State

Greg Law, co-founder & CTO
tl;dr

● Debugging dominates software development
  ○ Which means answering the question “what happened?”

● Record & replay is a new approach where the computer can just tell you

● Bugs can be fixed orders of magnitude more quickly

● Most software is not truly understood by anyone
In the beginning

Sir Maurice Wilkes, 1913-2010
In the beginning

I well remember [...] the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs.

Sir Maurice Wilkes, 1913-2010
Relays changed
Started Cosine Tape (Sine also)
Started Multi-Adder Test.

Nip Relay

First actual case of bug,
Antenna started,
closed down.

Relay #70
(moth) in relay.
Computers are hard
Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan
What happened?
What makes bugs really hard?

- Time between the root cause and effect being noticed
- Repeatability
What was the previous state?

Two options:

1. Save it.
2. Recompute it.

\[
\begin{align*}
a &= a + 1 & \checkmark \\
a &= b & \times
\end{align*}
\]
Snapshots

- Maintain snapshots through history
- Resume from these - run forward as needed
- Copy-on-Write for memory efficiency
- Adjust spacing to anticipate user’s needs
Event Log captures non-deterministic state

Event Log

Recorded during debug (or Live Recording)

Stored in memory

Replayed to reconstruct any point in history

Efficient, diff-based representation

Saved to create a recording file for later use
Undo Engine captures all non-determinism.

Some machine instructions are non-deterministic:

- rdtsc, cpuid, syscall, etc

Needs to capture all this and provide precise control over execution in general.

**Solution:** Runtime instrumentation.
In-process Virtualization

**Record**
- Captures all state changes in a running process
- 100% reproduction of execution history

**Replay**
- Replay & analyze the recording
- Detect root cause of bug
- Reverse debug and resolve
Multiple implementations

For Linux:
- Undo LiveRecorder (C++, Go, Java)
- rr (C++, Go)
- gdb process record

For Windows:
- Microsoft’s Time-Travel Debugger (C++, C#, Chakracore JS)
- RevDebug (C#, Java)
Works well in conjunction with live logging & tracing

Logging & tracing give a high-level ‘story’ of a program’s execution
Use it to know where to go in a recording
Apply logging to a recording
80/20 Rule
80/20 Rule
Business models / realisation models

Take off requires a lot of energy

- Open Source is hard to monetize
- Direct to developer is hard to get to critical mass
- Enterprise sales is hard to scale
1. Computers are hard & debugging is under-served
2. Record/replay is awesome
3. 80/20 rule does not always apply