LiveRecorder Data Sheet

Key insights

While Continuous Integration is now de facto standard practice (growing from 70% in 2015 to 88% in 2019),[1] software failures in test remain a major impediment to delivery speed, developer productivity, and software quality.

25-50%
of developer time is still spent debugging

82%
of software vendors have experienced issues in production related to a previously seen but unfixed test failure[2]

Make bugs 100% reproducible with time travel debugging

LiveRecorder provides a one-click workflow from a test failure to a time-travel debugger placed exactly at the point of failure – skipping the tedious steps usually required to reproduce the problem and enabling developers to start debugging test failures instantly.

Developers working on complex C/C++, Go, and Java software can now **save a huge amount of time diagnosing the root causes of new regressions, legacy bugs, and flaky tests**. Bugs that took days or weeks to fix can now be resolved in hours.

Record. Replay. Resolve.

RECORD

LiveRecorder captures a recording of an application failing under test (however intermittently), making bugs 100% reproducible.

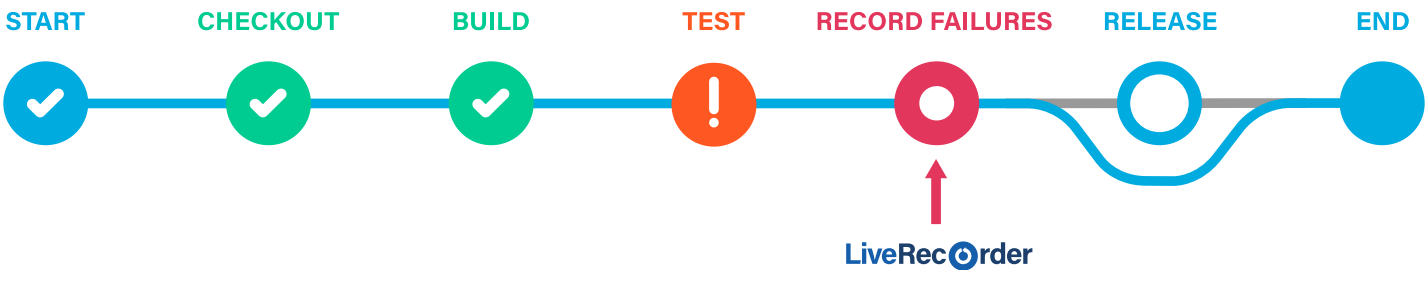
No time is wasted trying to reproduce test failures.

Expect a 2x to 5x slowdown against native execution. However, you only need to record once to get complete "video footage" of what happened, saving days or weeks of debugging time.

Note: there's also no need to record all tests in your suite, i.e. only record the tests that are failing.

100%
FAILURE REPRODUCIBILITY

Integrate LiveRecorder into your test suite to automatically generate recordings of any failed test.



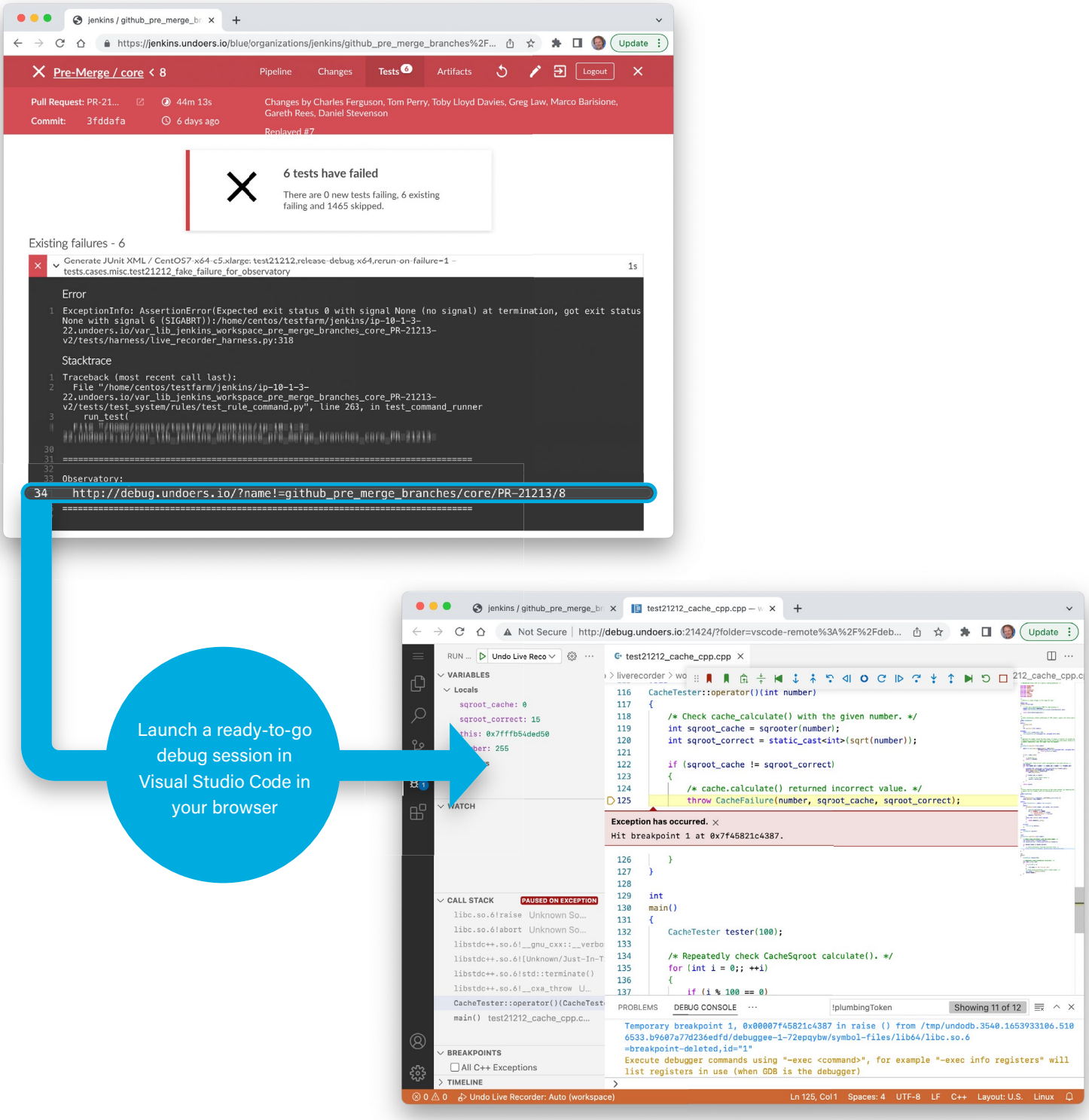
[1] Cambridge Judge Business School MBA Project Study, 2020
[2] Analyst firm study, Freeform Dynamics, 2018

LiveRecorder Data Sheet

REPLAY

Once a recording is generated, developers can jump from a test failure (or bug report) straight into a replay of the recording that captured the bug.

One-click workflow



Visual Studio Code opens up at the point of the crash

LiveRecorder Data Sheet

Portable recordings

Recordings are self-contained, i.e. they include all the non-deterministic inputs to the application including file and network inputs, signals, thread scheduling events, etc. This means that a recording contains all the information a developer needs in order to see what the process did, and why, down to individual machine instructions if required.

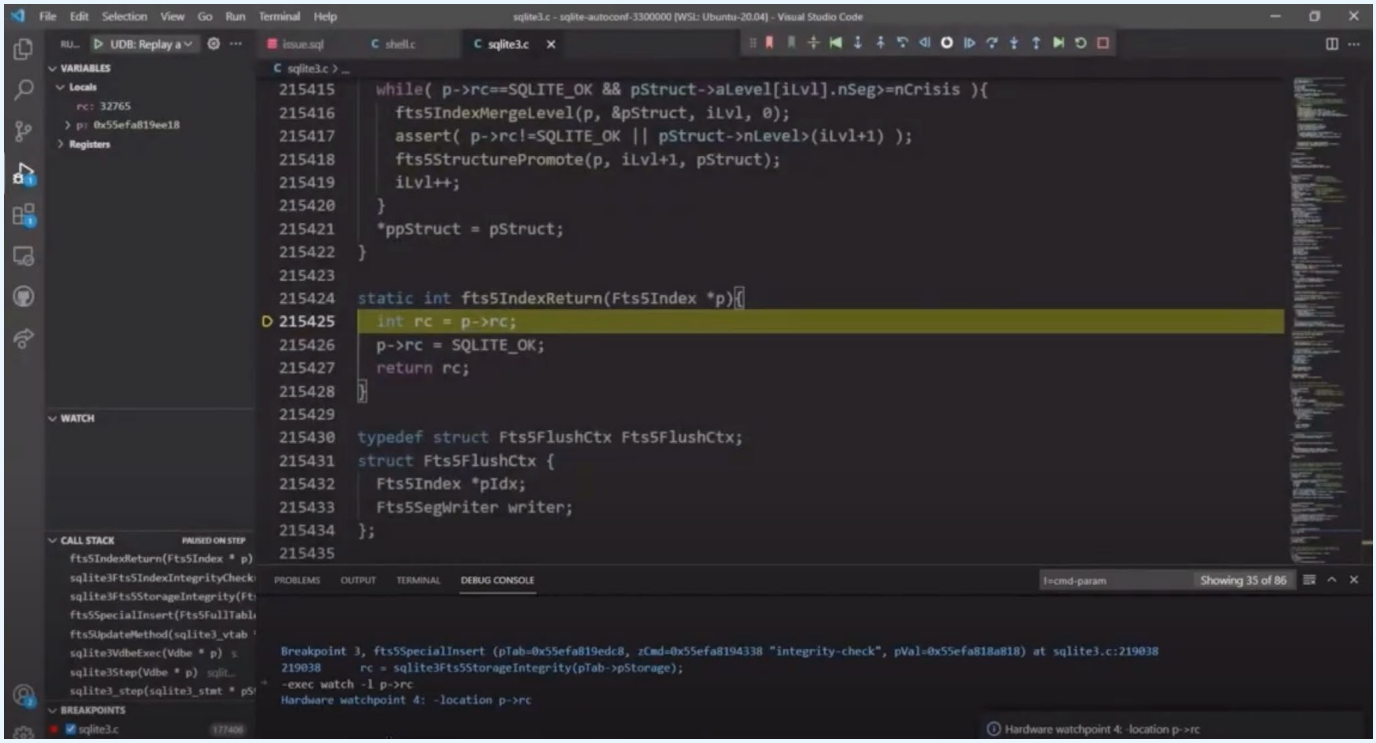
Recordings are also portable, i.e. they can be replayed anytime anywhere (out of the original environment and on a different CPU model).

RESOLVE

Bugs can easily be located by navigating recordings back and forth to see what the code actually did, and analyze internal program state at any point in time. Quickly trace program flow from bug manifestation all the way back to the root cause.

Time travel back in code execution and see what happened, when

- Replay the recording and inspect the complete state of the application – including the contents of all variables and the heap.
- Use the full range of debugger functionality to navigate the application's execution – stepping, running, breakpoints, watchpoints, catchpoints, etc. – but in reverse as well as forward (a.k.a. reverse debugging).



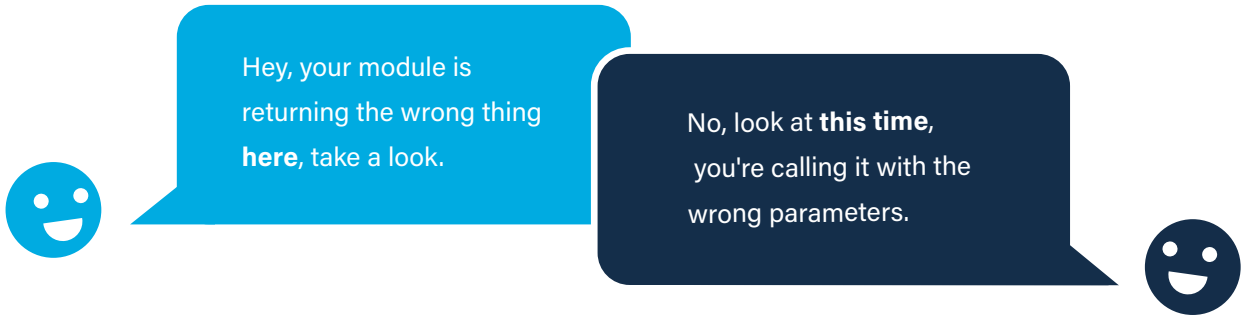
Watch reverse watchpoint in action

- Navigate through the application's execution history in ways simply not possible in a conventional debugger, e.g. spot a line of interest in a log file and jump inside a recording to the point in time and line of code when the application emitted that log line, or jump to the time a variable changed value or the application made a particular system call.

LiveRecorder Data Sheet

Asynchronous collaboration

Use recordings to collaborate remotely and asynchronously, and accelerate bug-fix time. Share links to moments in time with colleagues, share bookmarks, and add comments in recordings.



Ideal for complex software systems

Handles complex multi-threaded applications and those that use shared memory and asynchronous I/O.

Records program environments with multiple interacting processes. Multi-process correlation reveals the order in which processes and threads alter data structures in shared memory, and the order in which messages are passed between microservices.

Log Jump: pick a moment in time in the log file and jump to that moment in time in the recording.

Thread-fuzzing option randomizes thread execution to reveal race conditions and other multithreading defects (optional).

Recordings made within a container or VM are replayable on a physical machine, and vice versa.

Dynamic Logging: Log additional variables at replay-time to cater for "*I wish we'd logged that other variable*" moments. See what the output would have been without needing to re-run.

Integrations

Seamless integration into your Linux program and development workflow via command-line recording, API control, and IDE integrations (VS Code, IntelliJ IDEA, Eclipse, Clion, GoLand, and Emacs).

Integrates with any CI or Test orchestration system, including:

- Jenkins
- CircleCI
- TeamCity

Integrates with any issue/ticketing system, including:

- Jira
- GitHub
- Bugzilla

System requirements

Supports applications written in C/C++, Go, and Java, on Linux x86 Intel and AMD CPUs.

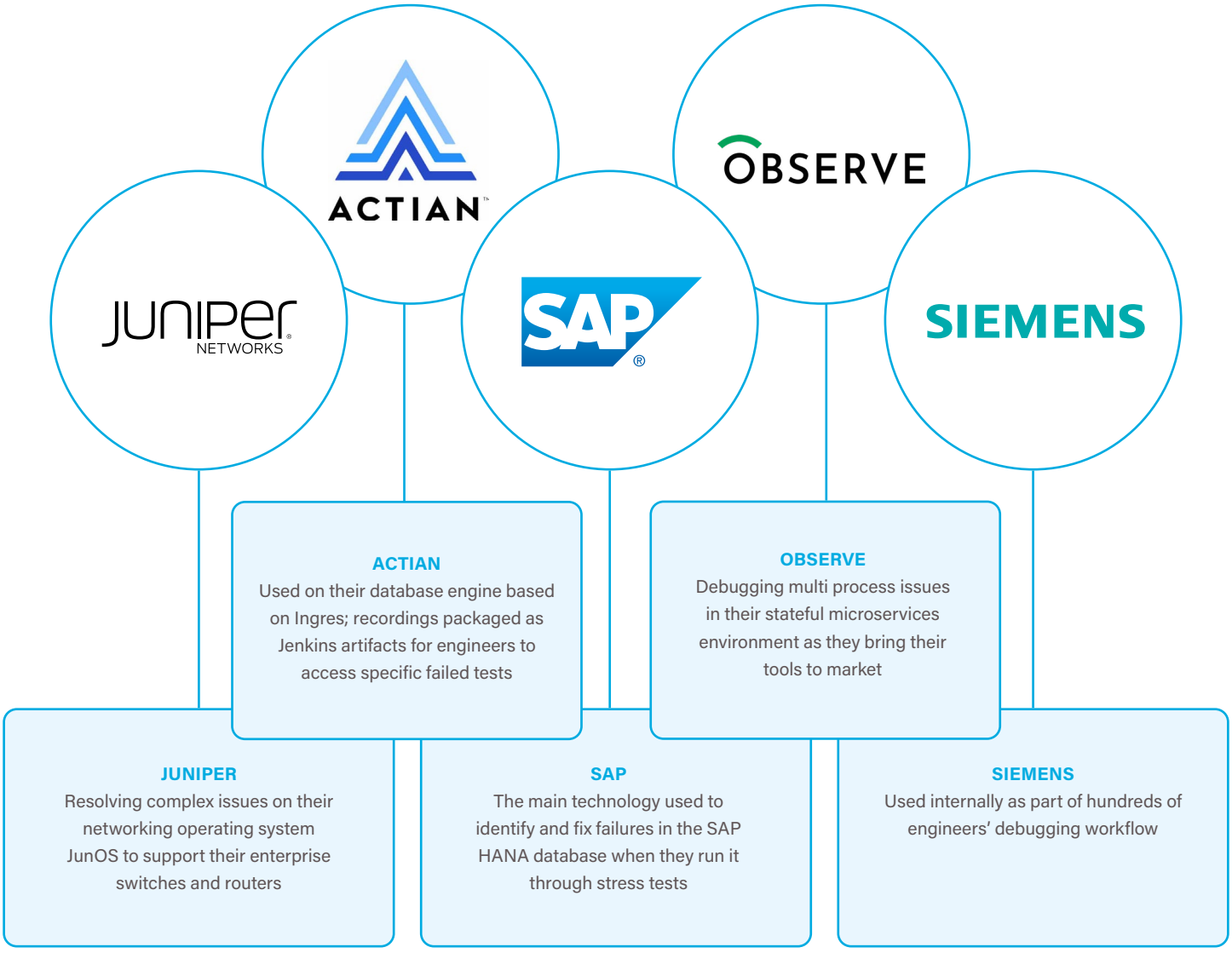
Supports applications running Linux kernel 3.10 or later. Compatible with all mainstream Linux distributions including:

- Red Hat Enterprise Linux and CentOS
- Fedora
- SUSE Linux Enterprise Server
- Ubuntu

LiveRecorder Data Sheet

Trusted by leaders

LiveRecorder is trusted by the world's leading enterprise software vendors.
Below are a handful of use cases.



About Undo

Undo is the time travel debugging company for Linux. We equip developers with the technology to understand complex code and fix bugs faster. Developers spend far too much time figuring out what code actually does – either to understand other people's code or to find and fix bugs. With time travel debugging, developers can see exactly what the software did, and create better software faster.

Visit undo.io for more information.