DYALOG
Eastbourne 2026

Testing the Untestable



GUIs, Fuzzers, and Dyalog APL

Aarush Bhat

Who am I?



Aarush Bhat

With Dyalog since Jan’23. I work on EWC, Testing and CITA



DYALOG

Glasgow 2024

ullu: Tests for Dyalog APL



Aarush Bhat

Dyalog Ltd.

DYALOG
Eastbourne 2026

EWC in 2026

Aarush Bhat

Dyalog Ltd.



Why testing matters

- A bug in the interpreter is a bug in every application built on it
- Every release ships to several platforms and editions; each is a chance for behaviour to differ
- Users never see the testing; they only see the release that did or did not break
- Most of the test suites run continuously, not just before a release

Testing a car



Engine
The language: interpreter, primitives, memory

Route
The environment: platforms and editions

Garage
The IDE and language tooling

Body
The APL tools and libraries we build

51 interpreters to test

- 7 platforms: Windows, Mac, Mac/Arm, AIX, Pi, Linux, Linux/Arm
- 4 versions, 7 platforms, 32/64-bit, Classic/Unicode
- That is… 51 interpreters of Dyalog APL, and every test suite must pass on each of them

Platforms and versions

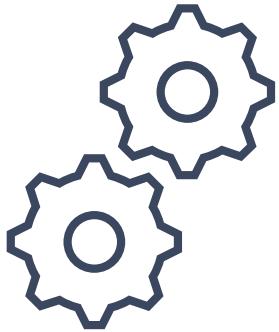
Platform	18.2	19.0	20.0	21.0	22.0
Windows	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U
macOS Intel	64U	64U	64U	–	–
macOS Apple Silicon	–	64U	64U	64U	64U
Linux Intel	64C 64U	64C 64U	64C 64U	64C 64U	64C 64U
Linux ARM	32U	32U	32U 64U	32U 64U	64U
AIX	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U	32C 32U 64C 64U

C Classic **U** Unicode 32 / 64: interpreter bit width – not supported

Agenda

- Testing the interpreter: unit tests, ullu, coverage, PQA, fuzzing
- CITA: testing our APL tools on real hardware
- Testing GUIs: `EWC` and `⎕WC`
- LLM agents as test writers
- Summary
- Questions

First, the engine



Engine

Interpreter, primitives, memory

- **QA workspaces**: one test per Mantis issue
- **ullu**: every primitive, every argument type
- **Coverage**: every line of C that runs
- **PQA**: performance statistics
- **Fuzzing**: AFL mutates one-line tests

QA workspaces

- Runs continuously against every interpreter build and every commit!
- Runs on our internal machines on Jenkins
- Checks results and behaviour rather than crashes
- Also, Shuffled! (∵) - 50,000x longer, parallel

One function per issue

`mNNNNN` returns 1 for pass, 0 for fail

Workspaces run in turn

The last, `miscLast`, runs the unit tests and prints `QAs_completed` for the nightly build

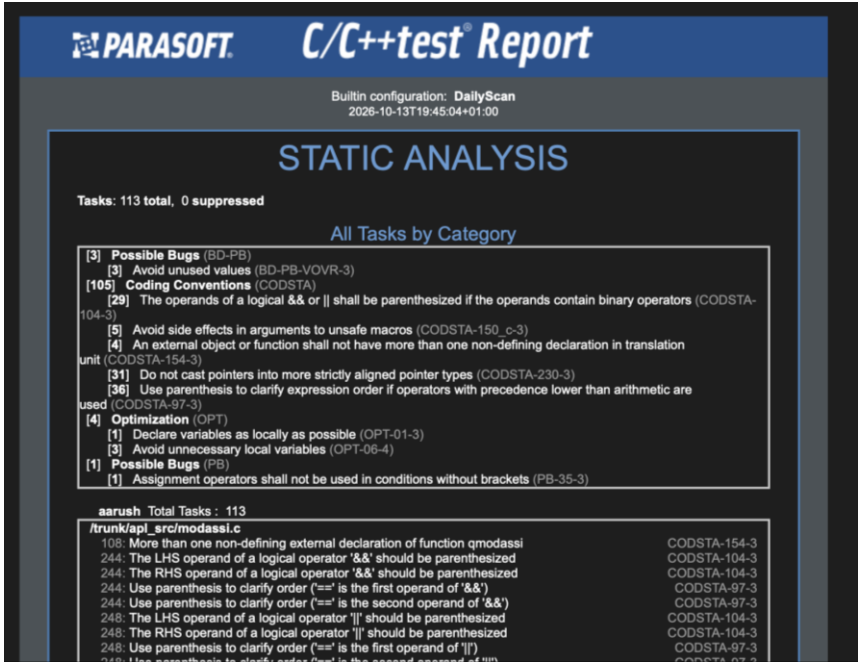
In-line Unit tests

- aka. doctests
- Inline with the source code
- Tests with basic one-line expressions, get picked up by the QA workspace

```
//⍙ (↑⍬)≡⍬
//⍙ (↑⍬⍬)≡2 0⍴0
//⍙ (↑⍬'')≡2 0⍴0
//⍙ (↑''⍬)≡2 0⍴''
//⍙ (↑(⊂1 2)3)≡(1 2)3
//⍙ (↑(⊂⊂1 2)3)≡(⊂1 2)3
//⍙ (↑2 3 0⍴0)≡2 3 0⍴0
```

Static Analysis

- Parasoft C/C++ test scans the interpreter's C source every day
- Flags possible bugs, coding convention breaks and optimizations



ullu

- One primitive at a time, across argument types, ranks and shapes
- Use modelled function to verify correctness
- Tries to catch cases with data hand-written test would never think to write
- A 2nd coat of paint



u

Model functions

- Unique R←∪Y
  - `{0=≢⍵:⍵ ⋄ ↑,⊃{⍺,(∧/⍺≢¨⍵)/⍵}⍨⌸/⍸⊂⍨¨⊂⍤¯1⊢⍵}`
- Add R←X+Y
  - `{⍺--⍵}`
- Magnitude R←|Y
  - `{⍵×(¯1@(∊∘0)(⍵>0))}`



u

Tests

- Unique R←∪Y
  - `Assert (≢data)≥≢∪data`
- Add R←X+Y
  - `RunVariations (2×data) data data`
- Magnitude R←|Y
  - `RunVariations (modelMag data) data`



Coverage builds

- Records every line of C source that runs
- Each suite is run separately, so we know what all tests reach
- The gaps tell us where to write the next test, and which code no suite has ever touched

Covera

- Records
- Each sui
- The gap
  no suite

```
16    switch (mult)
16    {
 0    case 1: b_times_scal(bound, type);
 0       break;
 2    case 2: if (Larg==Bool0)
 2          zero_each(bound);
 2       else
 2          Rslt = zap(Rarg);
 2       break;
14    case 3: calc_rslt(type, type, APLBOOL, bound,
14             mulbsfns[type-APLSINT]);
14       break;
16    }
```

Covera

- Records

```
16    switch (mult)
16    {
 0    case 1: b_times_scal(bound, type);
 0       break;
```

	Function Coverage	Line Coverage	Region Coverage	Branch Coverage
src/same_ibeam.c	66.67% (2/3)	39.73% (29/73)	38.67% (58/150)	32.00% (16/50)
src/same_ibeam.h	100.00% (2/2)	100.00% (6/6)	100.00% (2/2)	- (0/0)
src/same_template.h	100.00% (3/3)	86.05% (259/301)	53.18% (643/1209)	45.33% (340/750)
src/scalar.h	100.00% (1/1)	82.35% (14/17)	100.00% (4/4)	100.00% (2/2)
src/scalarscalar.cpp	92.16% (94/102)	83.90% (469/559)	80.72% (1394/1727)	82.71% (220/266)
src/scald.cpp	100.00% (36/36)	98.61% (995/1009)	96.88% (1953/2016)	93.40% (835/894)
src/scalm.cpp	95.35% (123/129)	97.17% (789/812)	96.33% (1522/1580)	89.72% (506/564)
src/scan.c	100.00% (60/60)	95.74% (1820/1901)	93.34% (1304/1397)	89.33% (804/900)

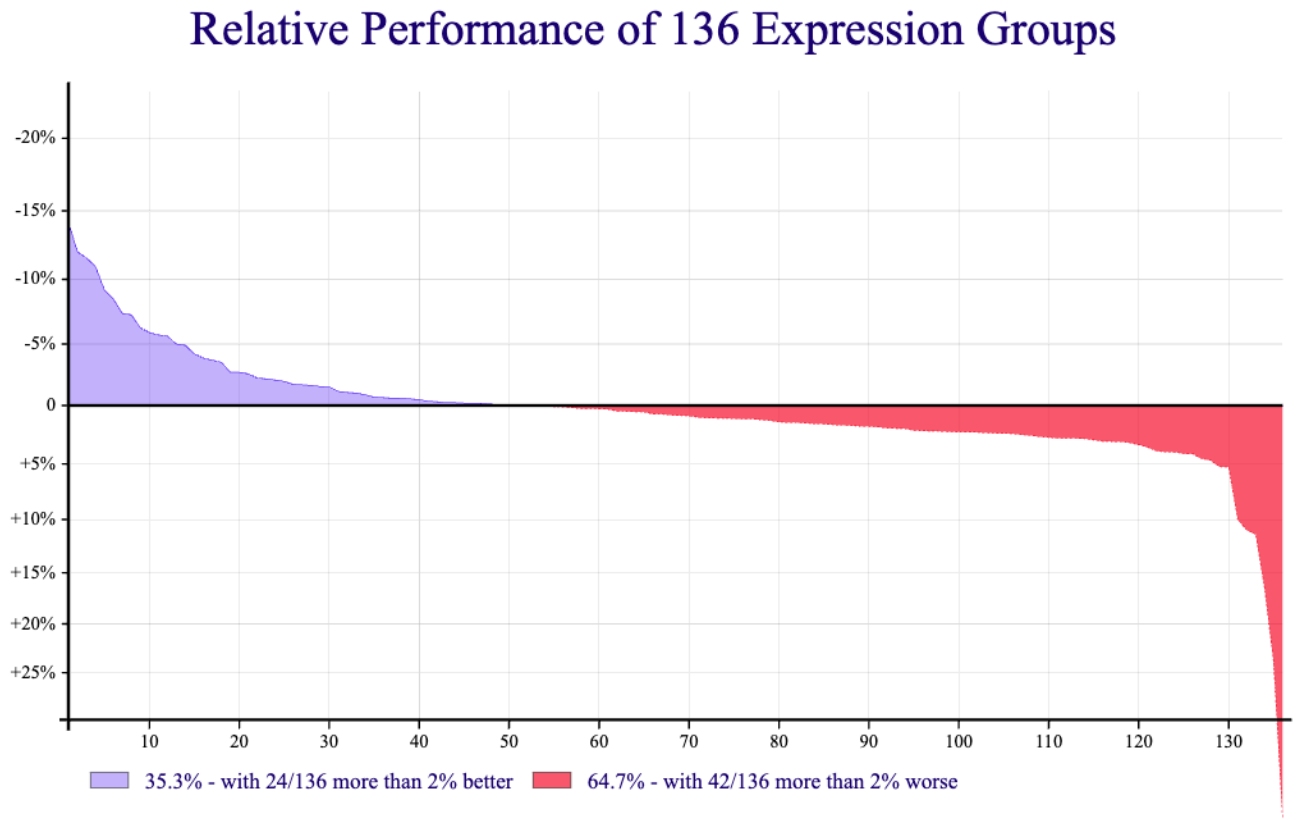
```
14    case 3: calc_
14          mulbsfns[type-APLSINT]);
14       break;
16    }
```

pqa aka. Performance QA

- Collects performance statistics
- Several thousand APL expressions
- Compares performance of APL interpreters.

Performance Comparison
Between Windows-64 19.0.49285.0 W Development and Windows-64 18.2.49285.0 W Development

Geometric mean of 136 expression groups: +0.6%

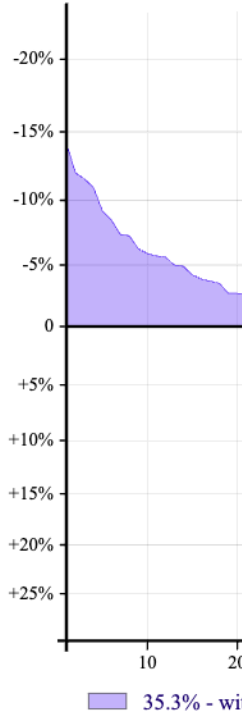


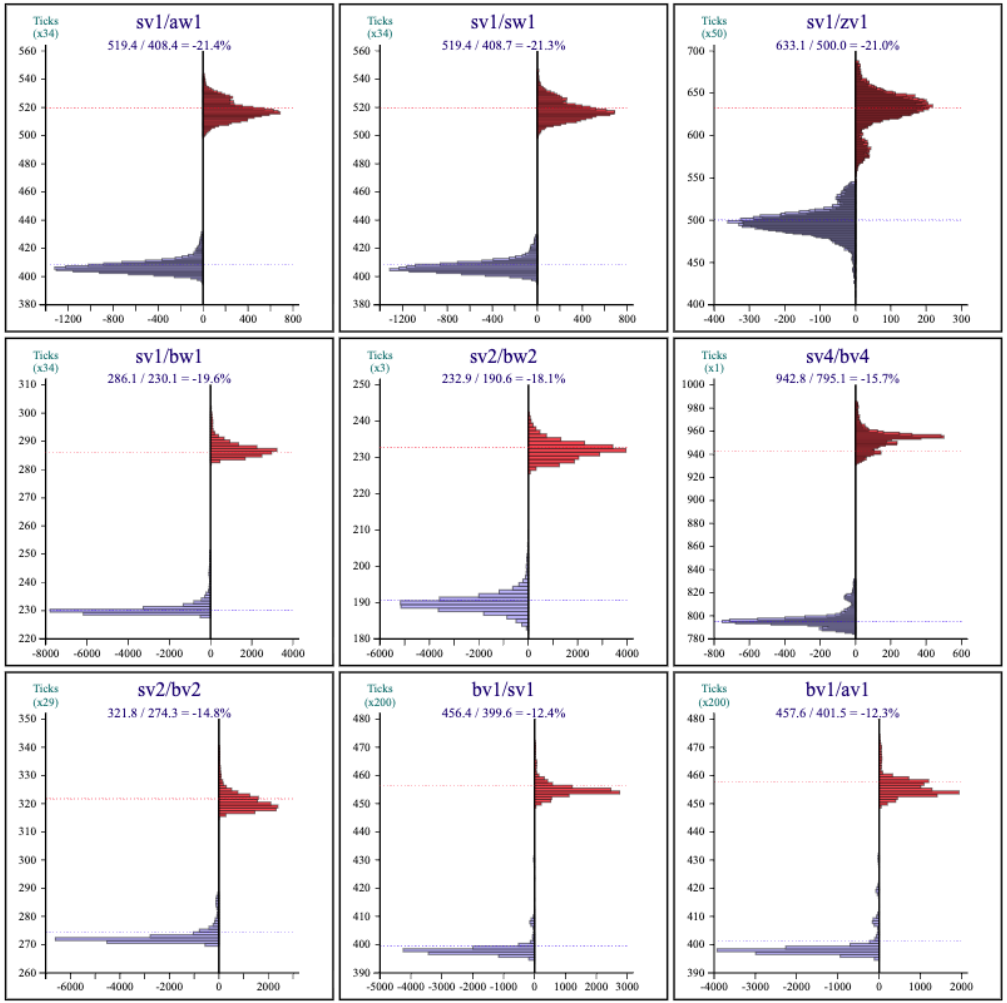
Best Expressions

Between Win

Development

Re





Fuzzing

- AFL mutates one-line APL tests taken from the unit tests
- American Fuzzy Lop (AFL – not a rabbit)
- Runs unattended for hours/days at a time
- APL errors are expected; only crashes and hangs count

~3.6 trillion

expressions tried so far

47,000

kept as “interesting expressions”

Fuzzing

- APL tests taken

```
#.(f←∘⊢)¨0
```

~3.6 trillion

```
3 1{⍵≡⍺⍺{+/,⍵}⌹⍺⊢⍵}⍤0 0
```

- American Fuzzy Lop (AFL – not a rabbit)

47,000

```
'⍞R(')⍞R'⍣)'
```

- Ru s/days at a time

```
#.{⍵}.(# #).{⍵}
```

- APL error hes and hangs

```
(,#).(@0)∘⊢⍬
```

Fuzzing

- APL tests taken … n

  … r

One of the longest ones Andy has seen recently!

```
{0::⎕FX'⊆←'⋄≡f{⍵}{0::0=⎕FX'r←f'  'r←⍬'
':for⊂t:in≡⊢∘≢⌸⍴0⋄:trap{≤&/⍵0}0⋄:end⋄:trap 0⋄{2}0⋄:end⍝⍳⊆←'
':end'⋄f}¨&0 0}¨⍳6
```

- APL error… hes and hangs

```
(,#).(@0)∘⊢⍬
```

Crashes become tests

Crash

fuzzer finds an input that kills the interpreter

Minimise

cut the expression to the smallest form that still crashes

Deduplicate

drop repeats of the same minimised input

Add to corpus

a growing body of crash tests

Replay

every platform and edition, on every later build

CITA

- **C**ontinuous **I**ntegration **T**esting for **A**PL Tools
- Windows 11, Linux x86, Linux ARM, Docker
- Runs on every PR, nightly for every repo, or on demand
- Bug somewhere could be a bug anywhere
- Tests the body and the engine with it

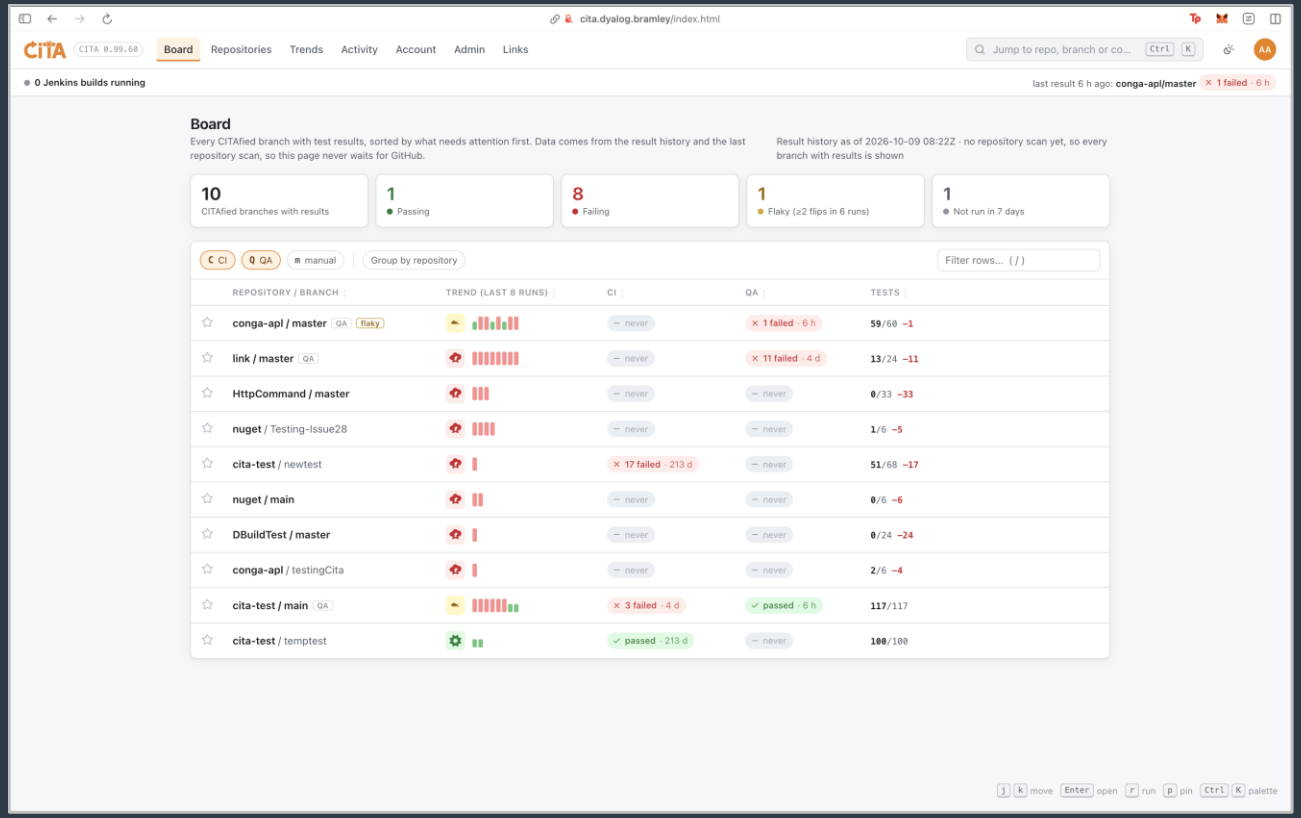
The matrix

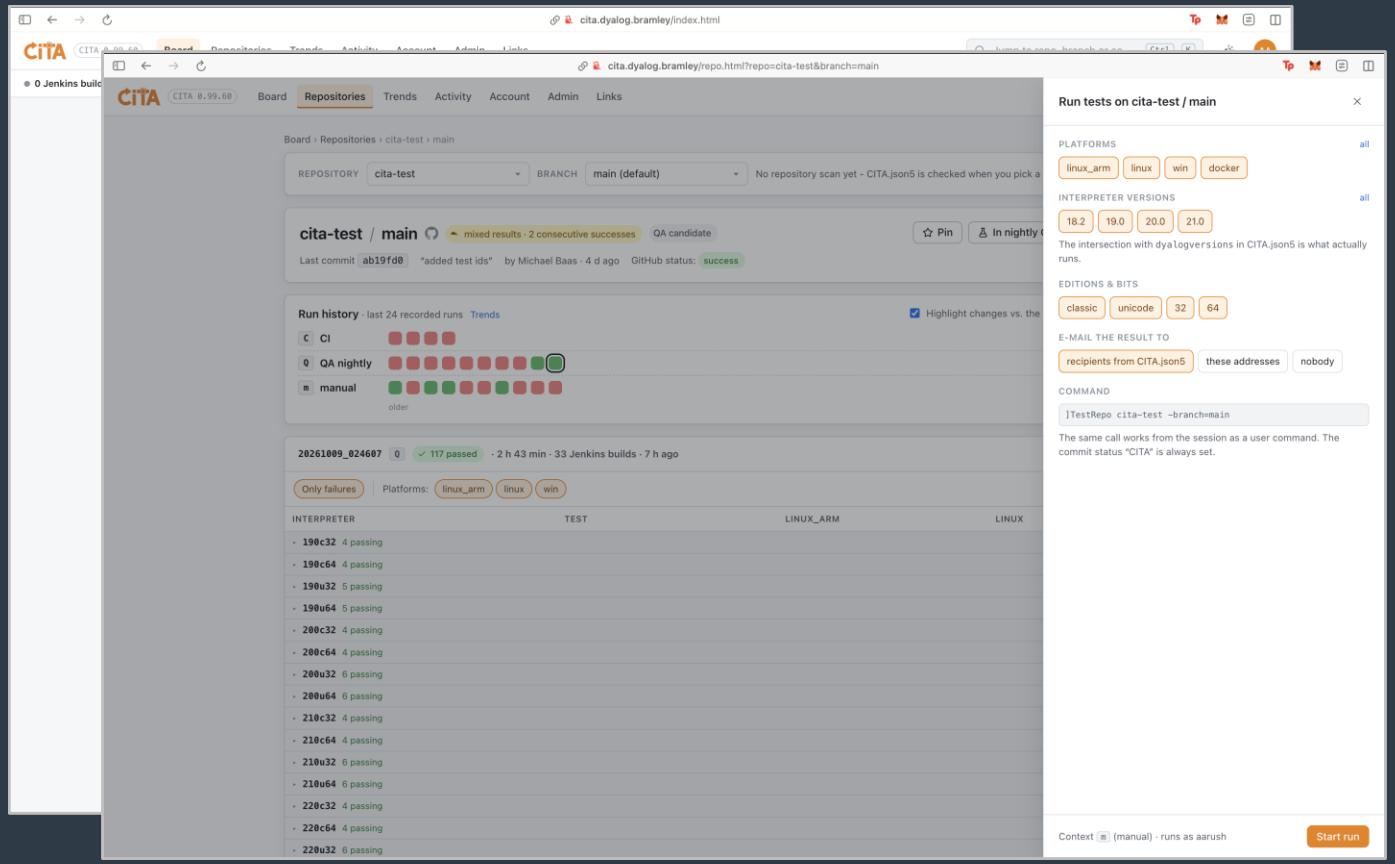
Versions 18.2 to 21.0

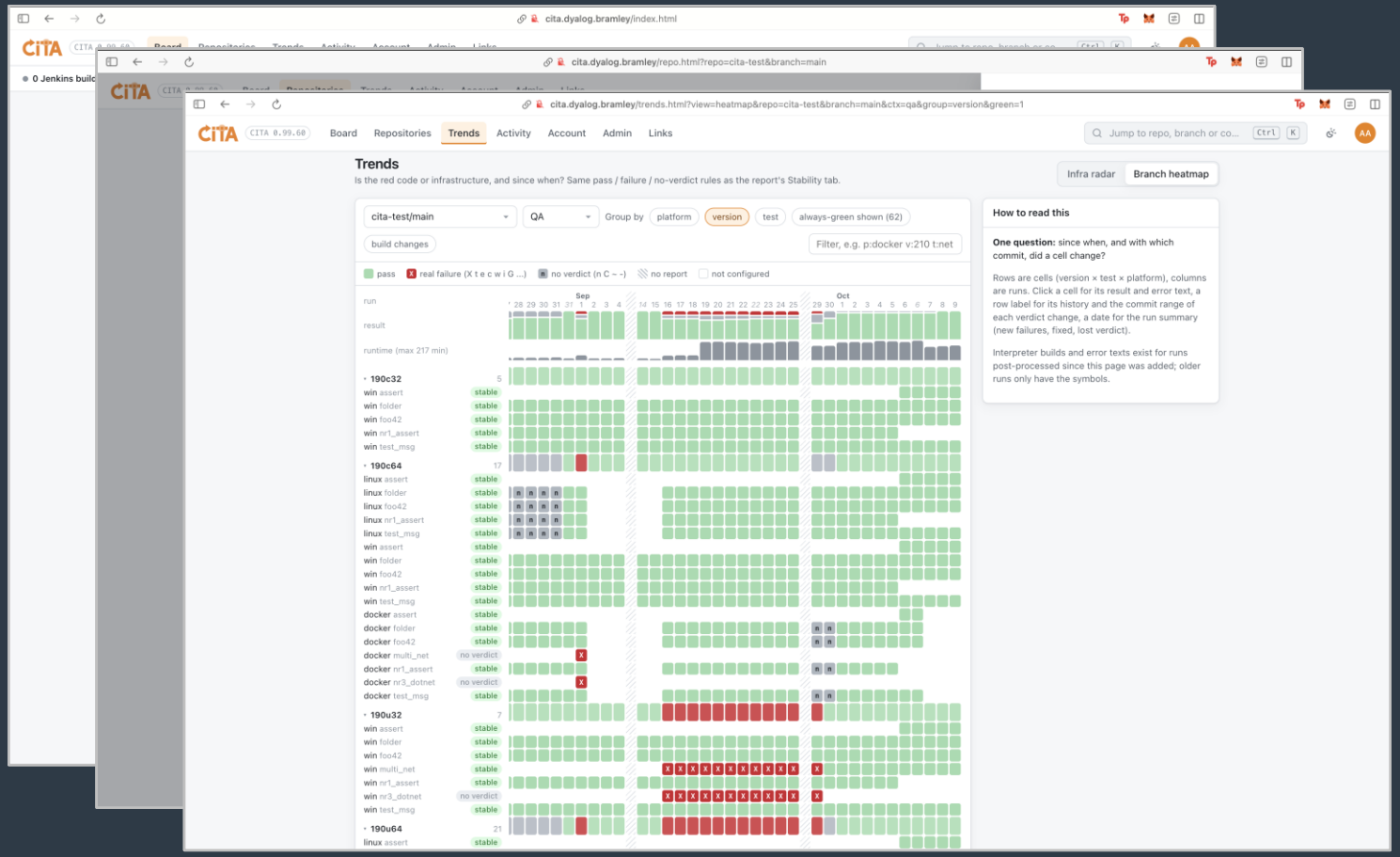
Classic and Unicode

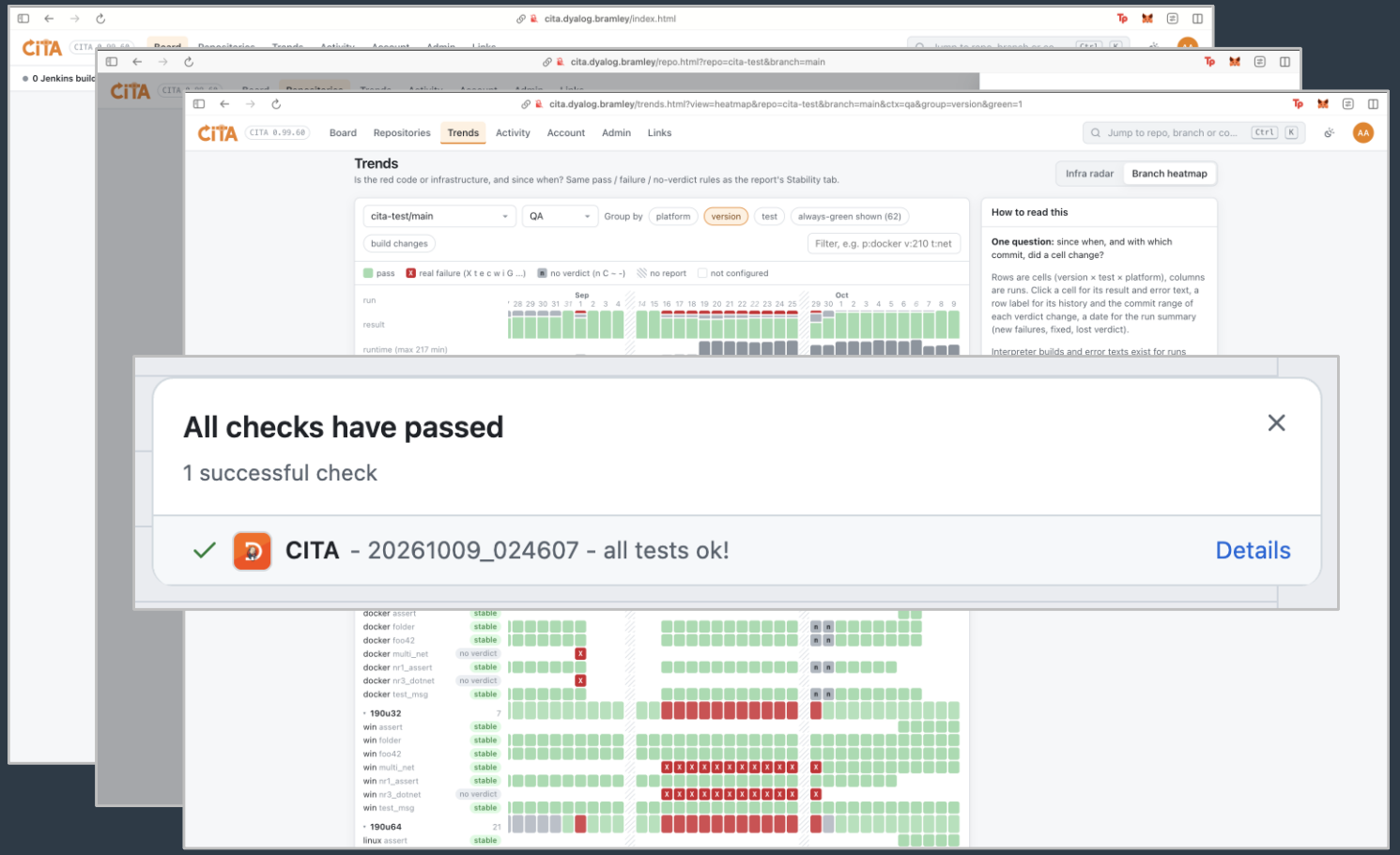
32- and 64-bit on Windows

.NET: none, 4.8, 8.0, 10.0


cita.dyalog.bramley/index.html
CITA
CITA 0.99.60
Board
Repositories
Trends
Activity
Account
Admin
Links
Jump to repo, branch or co...
Ctrl K
AA
0 Jenkins builds running
last result 6 h ago: conga-apl/master
1 failed · 6 h
Board
Every CITAfied branch with test results, sorted by what needs attention first. Data comes from the result history and the last repository scan, so this page never waits for GitHub.
Result history as of 2026-10-09 08:22Z · no repository scan yet, so every branch with results is shown
10
CITAfied branches with results
1
Passing
8
Failing
1
Flaky (≥2 flips in 6 runs)
1
Not run in 7 days
C CI
Q QA
manual
Group by repository
Filter rows... (/)
REPOSITORY / BRANCH
TREND (LAST 8 RUNS)
CI
QA
TESTS
conga-apl / master QA flaky
never
1 failed · 6 h
59/60 −1
link / master QA
never
11 failed · 4 d
13/24 −11
HttpCommand / master
never
never
0/33 −33
nuget / Testing-Issue28
never
never
1/6 −5
cita-test / newtest
never
17 failed · 213 d
never
51/68 −17
nuget / main
never
never
0/6 −6
DBuildTest / master
never
never
0/24 −24
conga-apl / testingCita
never
never
2/6 −4
cita-test / main QA
3 failed · 4 d
passed · 6 h
117/117
cita-test / temptest
passed · 213 d
never
100/100
j k move
Enter open
r run
p pin
Ctrl K palette


cita.dyalog.bramley/index.html
cita.dyalog.bramley/repo.html?repo=cita-test&branch=main
CITA
CITA 0.99.60
Board
Repositories
Trends
Activity
Account
Admin
Links
Board › Repositories › cita-test › main
REPOSITORY
cita-test
BRANCH
main (default)
No repository scan yet – CITA.json5 is checked when you pick a
cita-test / main
mixed results · 2 consecutive successes
QA candidate
Pin
Last commit ab19fd0 "added test ids" by Michael Baas · 4 d ago GitHub status: success
Run history · last 24 recorded runs Trends
Highlight changes vs. the
C CI
Q QA nightly
m manual
older
20261009_024607 Q ✓ 117 passed · 2 h 43 min · 33 Jenkins builds · 7 h ago
Only failures
Platforms: linux_arm linux win
INTERPRETER
TEST
LINUX_ARM
LINUX
190c32 4 passing
190c64 4 passing
190u32 5 passing
190u64 5 passing
200c32 4 passing
200c64 4 passing
200u32 6 passing
200u64 6 passing
210c32 4 passing
210c64 4 passing
210u32 6 passing
210u64 6 passing
220c32 4 passing
220c64 4 passing
220u32 6 passing
Run tests on cita-test / main
PLATFORMS
all
linux_arm
linux
win
docker
INTERPRETER VERSIONS
all
18.2
19.0
20.0
21.0
The intersection with ⎕VERSIONS in CITA.json5 is what actually runs.
EDITIONS & BITS
classic
unicode
32
64
E-MAIL THE RESULT TO
recipients from CITA.json5
these addresses
nobody
COMMAND
]TestRepo cita-test -branch=main
The same call works from the session as a user command. The commit status "CITA" is always set.
Context m (manual) · runs as aarush
Start run


cita.dyalog.bramley/index.html
cita.dyalog.bramley/repo.html?repo=cita-test&branch=main
cita.dyalog.bramley/trends.html?view=heatmap&repo=cita-test&branch=main&ctx=qa&group=version&green=1
CITA
CITA 0.99.60
Board
Repositories
Trends
Activity
Account
Admin
Links
Jump to repo, branch or co...
Ctrl
K
AA
Trends
Is the red code or infrastructure, and since when? Same pass / failure / no-verdict rules as the report's Stability tab.
Infra radar
Branch heatmap
cita-test/main
QA
Group by
platform
version
test
always-green shown (62)
build changes
Filter, e.g. p:docker v:210 t:net
pass
real failure (X t e c w i G ...)
no verdict (n C ~ -)
no report
not configured
Sep
Oct
run
result
runtime (max 217 min)
190c32
win assert
win folder
win foo42
win nr1_assert
win test_msg
190c64
linux assert
linux folder
linux foo42
linux nr1_assert
linux test_msg
win assert
win folder
win foo42
win nr1_assert
win test_msg
docker assert
docker folder
docker foo42
docker multi_net
docker nr1_assert
docker nr3_dotnet
docker test_msg
190u32
win assert
win folder
win foo42
win multi_net
win nr1_assert
win nr3_dotnet
win test_msg
190u64
linux assert
stable
no verdict
How to read this
One question: since when, and with which commit, did a cell change?
Rows are cells (version × test × platform), columns are runs. Click a cell for its result and error text, a row label for its history and the commit range of each verdict change, a date for the run summary (new failures, fixed, lost verdict).
Interpreter builds and error texts exist for runs post-processed since this page was added; older runs only have the symbols.


cita.dyalog.bramley/index.html
cita.dyalog.bramley/repo.html?repo=cita-test&branch=main
cita.dyalog.bramley/trends.html?view=heatmap&repo=cita-test&branch=main&ctx=qa&group=version&green=1
CITA
CITA 0.99.60
Board
Repositories
Trends
Activity
Account
Admin
Links
Jump to repo, branch or co...
Ctrl
K
AA
Trends
Is the red code or infrastructure, and since when? Same pass / failure / no-verdict rules as the report's Stability tab.
Infra radar
Branch heatmap
cita-test/main
QA
Group by
platform
version
test
always-green shown (62)
build changes
Filter, e.g. p:docker v:210 t:net
pass
real failure (X t e c w i G ...)
no verdict (n C ~ -)
no report
not configured
run
result
runtime (max 217 min)
Sep
Oct
How to read this
One question: since when, and with which commit, did a cell change?
Rows are cells (version × test × platform), columns are runs. Click a cell for its result and error text, a row label for its history and the commit range of each verdict change, a date for the run summary (new failures, fixed, lost verdict).
Interpreter builds and error texts exist for runs
All checks have passed
1 successful check
CITA - 20261009_024607 - all tests ok!
Details
docker assert
docker folder
docker foo42
docker multi_net
docker nr1_assert
docker nr3_dotnet
docker test_msg
190u32
win assert
win folder
win foo42
win multi_net
win nr1_assert
win nr3_dotnet
win test_msg
190u64
linux assert
stable
no verdict

Testing GUIs

- A GUI test has to click, type, wait and look
- “Looks right” is hard to assert
- Tests are slow to write and break whenever the UI changes, so most teams give up and test by hand
- We can test ⬚WC GUIs in a similar way, using `pywinauto`

How we test EWC

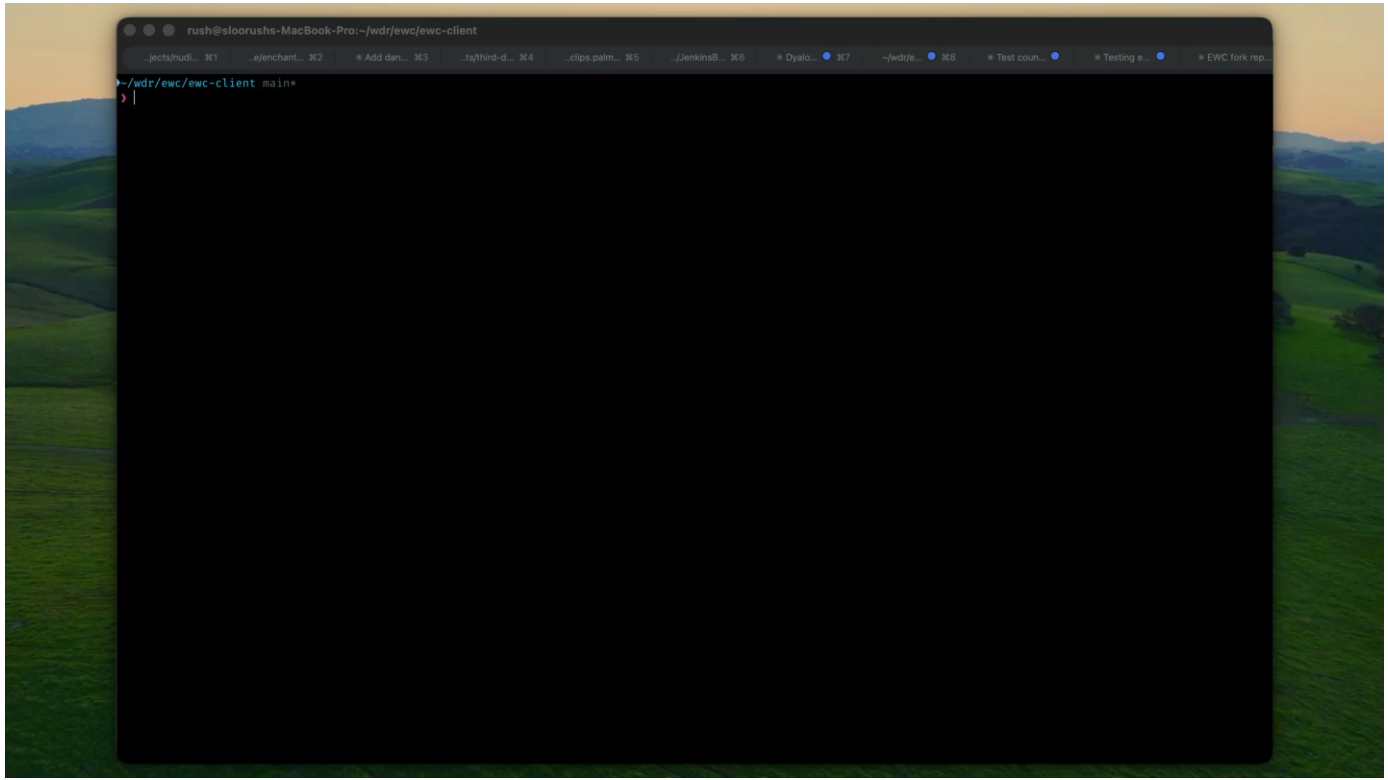
Real stack, no mocks

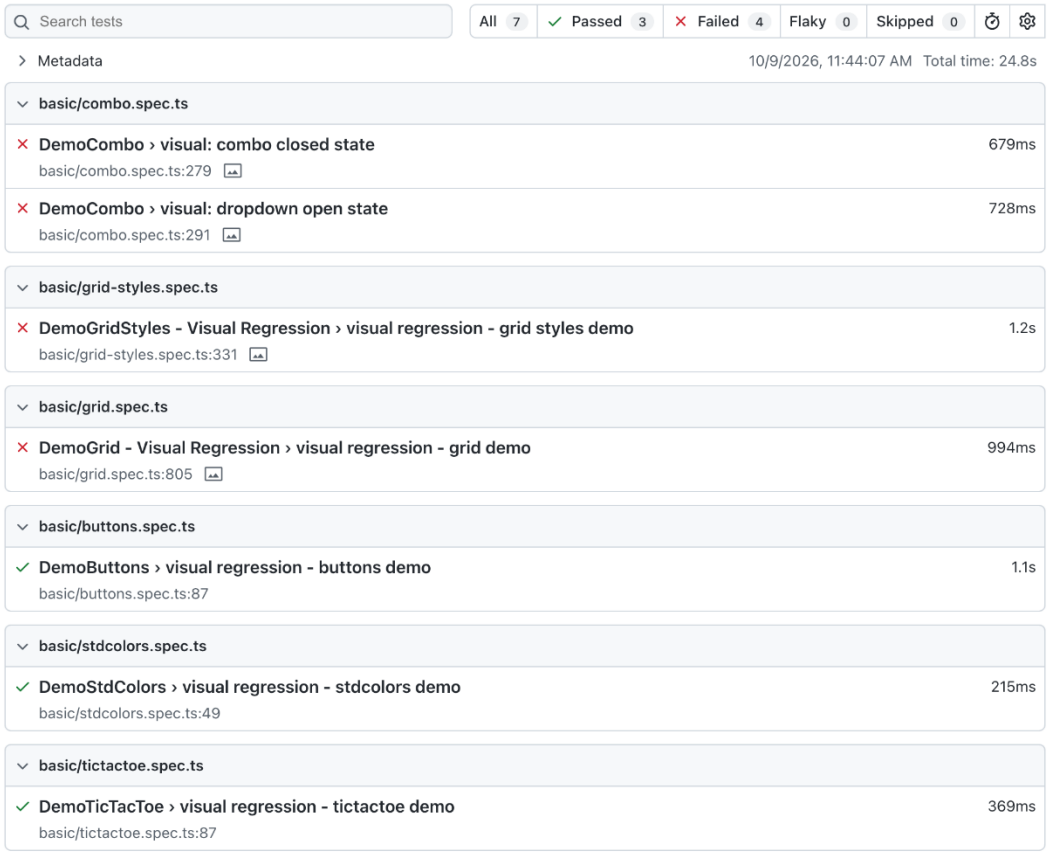
- Real browser/desktop
- ~100 demo apps as fixtures
- Dyalog in Docker, locally and in CI

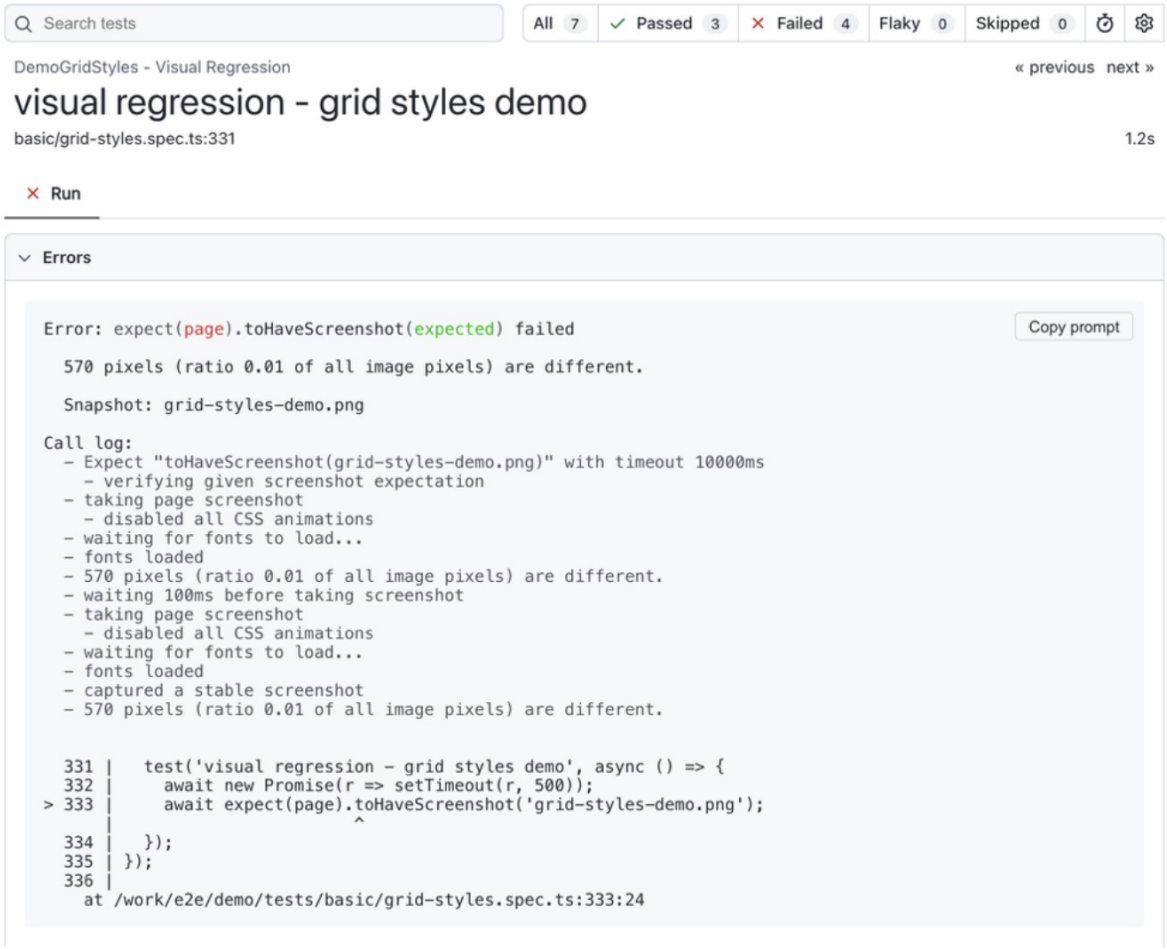
What runs

- Automated CI on GitHub
- Every commit is tested
- ~445 Playwright tests
- ~50 test Uis
- Applications tests running on local – soon to be moved to a CI

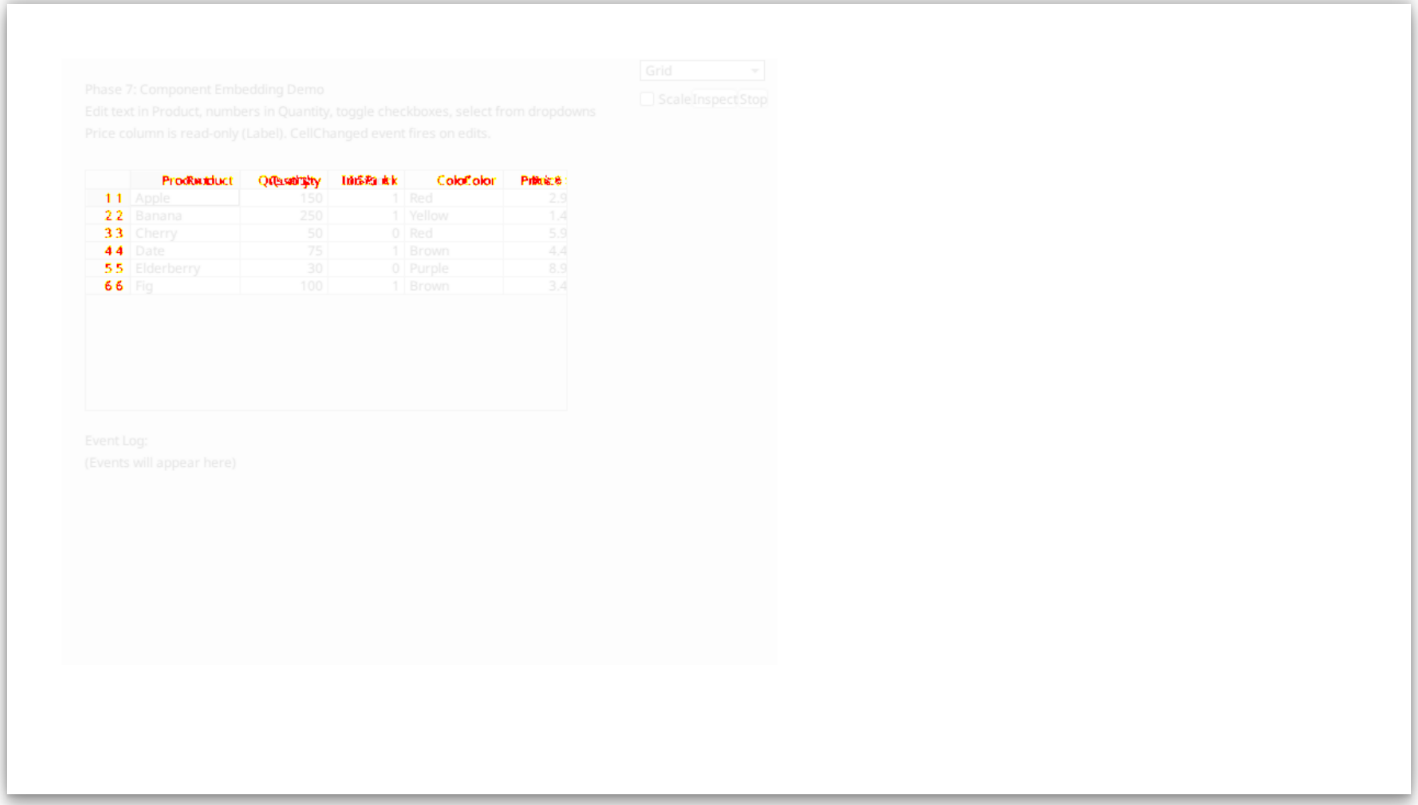
EWC tests in action



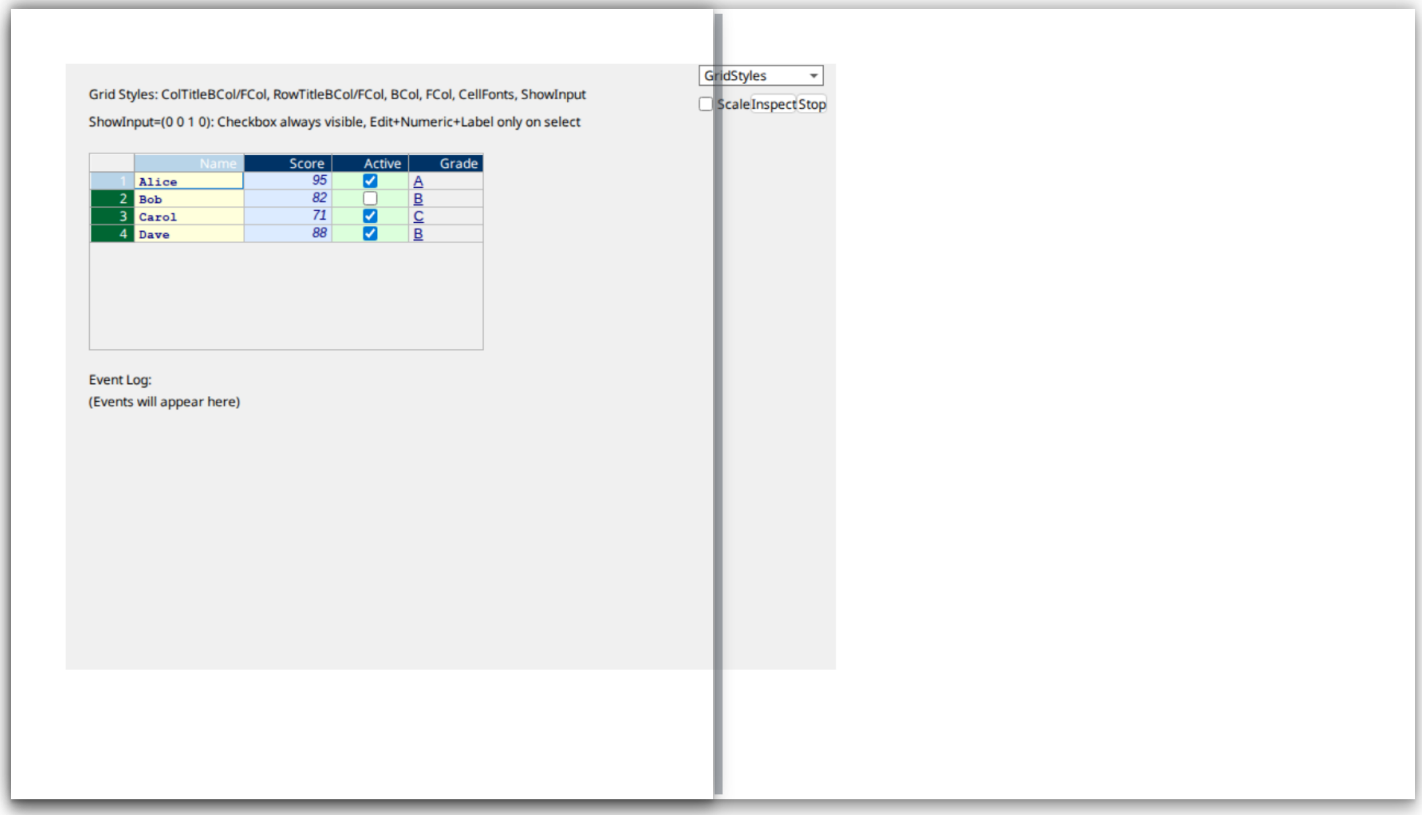

Search tests
All 7
Passed 3
Failed 4
Flaky 0
Skipped 0
Metadata
10/9/2026, 11:44:07 AM Total time: 24.8s
basic/combo.spec.ts
DemoCombo › visual: combo closed state
679ms
basic/combo.spec.ts:279
DemoCombo › visual: dropdown open state
728ms
basic/combo.spec.ts:291
basic/grid-styles.spec.ts
DemoGridStyles - Visual Regression › visual regression - grid styles demo
1.2s
basic/grid-styles.spec.ts:331
basic/grid.spec.ts
DemoGrid - Visual Regression › visual regression - grid demo
994ms
basic/grid.spec.ts:805
basic/buttons.spec.ts
DemoButtons › visual regression - buttons demo
1.1s
basic/buttons.spec.ts:87
basic/stdcolors.spec.ts
DemoStdColors › visual regression - stdcolors demo
215ms
basic/stdcolors.spec.ts:49
basic/tictactoe.spec.ts
DemoTicTacToe › visual regression - tictactoe demo
369ms
basic/tictactoe.spec.ts:87


Search tests
All 7
Passed 3
Failed 4
Flaky 0
Skipped 0
DemoGridStyles - Visual Regression
« previous next »
visual regression - grid styles demo
basic/grid-styles.spec.ts:331
1.2s
Run
Errors
Copy prompt
Error: expect(page).toHaveScreenshot(expected) failed
570 pixels (ratio 0.01 of all image pixels) are different.
Snapshot: grid-styles-demo.png
Call log:
- Expect "toHaveScreenshot(grid-styles-demo.png)" with timeout 10000ms
- verifying given screenshot expectation
- taking page screenshot
- disabled all CSS animations
- waiting for fonts to load...
- fonts loaded
- 570 pixels (ratio 0.01 of all image pixels) are different.
- waiting 100ms before taking screenshot
- taking page screenshot
- disabled all CSS animations
- waiting for fonts to load...
- fonts loaded
- captured a stable screenshot
- 570 pixels (ratio 0.01 of all image pixels) are different.
331 | test('visual regression - grid styles demo', async () => {
332 | await new Promise(r => setTimeout(r, 500));
> 333 | await expect(page).toHaveScreenshot('grid-styles-demo.png');
| ^
334 | });
335 | });
336 |
at /work/e2e/demo/tests/basic/grid-styles.spec.ts:333:24

Diff Actual Expected Side by side Slider



Diff Actual Expected Side by side **Slider**



Visual baselines and CI

Pixel baselines

- **≤ 0.01 pixels** may differ
- **A human** approves changes to baselines
- Helps **check inconsistencies** visually

Every PR

- **5 shards**, fresh Dyalog each
- Different suite in **multi-mode**
- **Failures** shown on the PR

Agents write and drive the tests

- **Claude Code** on the real stack
- Wrote much of **~445 tests**
- **Reproduces bugs,** reviews code
- **Humans** own the truth!

Agents run the stack, humans sign off

Run it

- **Dyalog** in Docker headed or headless
- **Playwright**, report, screenshots – for humans to verify
- **Parallel** containers per suite

Check it

- **Side by side** with native `⎕WC`
- **Dyalog's documented** semantics for `⎕WC`

Benefits

- **Fast** to write
- **Cheap** to fix
- **Wide** coverage
- **Baselines** need a human

What agents learn is written down



CLAUDE.md files

How the system works, how to run tests, baseline rules



Agent memory

Hard-won facts, e.g. “baselines are Linux-only”



How to run?

Agents remember how to run the relevant tests for the problem

What this changes

- Less time writing tests, more time thinking what's correct
- Updating tests is easy and quick
- No small 2px changes sneak up on us
- The cost of **testing the untestable** drops enough that GUIs get covered like everything else

Summary



Unit tests

Factory checklist

Every part checked off the line



ullu/PQA

Dyno

Each primitive run under load



Coverage

Inspection sheet

Which parts did the test drive touch?



Fuzzing

Crash test

Drive like a maniac to find what breaks



CITA

Test routes

Every road surface, on real tarmac



GUI + visual regression

Driver's-eye camera

Same dashboard on every build?



LLM agents

New test driver

Never seen the car, pokes every button

Questions

> *"It is not just about finding things wrong with Dyalog APL, but about making us more confident when making enhancements."*
>
> – me ☺

Testing your own applications, or ideas for how we test Dyalog APL? Get in touch

aarush@dyalog.com