



Rapita Systems Ltd

News and updates

Acquisition – April 2016

Rapita acquired by Danlaw Inc.

Expansion:

- Larger US operation
- Provide additional services/consultancy
- Increase automotive support

- We are growing and hiring (*hint*)
- Most things unchanged
 - UK company, operating from York
 - Management team (Guillem Bernat, Ian Broster, Antoine Colin)



What Rapita does

Critical software verification and testing *solutions*
Reducing the cost of software verification

- Tools
 - Tim
 - Coc
 - Sys
- Ada, C

Integrate with your ...
... Code
... Process
... Needs

- Good at the really big and hard embedded projects
 - (as well as the small and on-host ones!)

Other Rapita news....

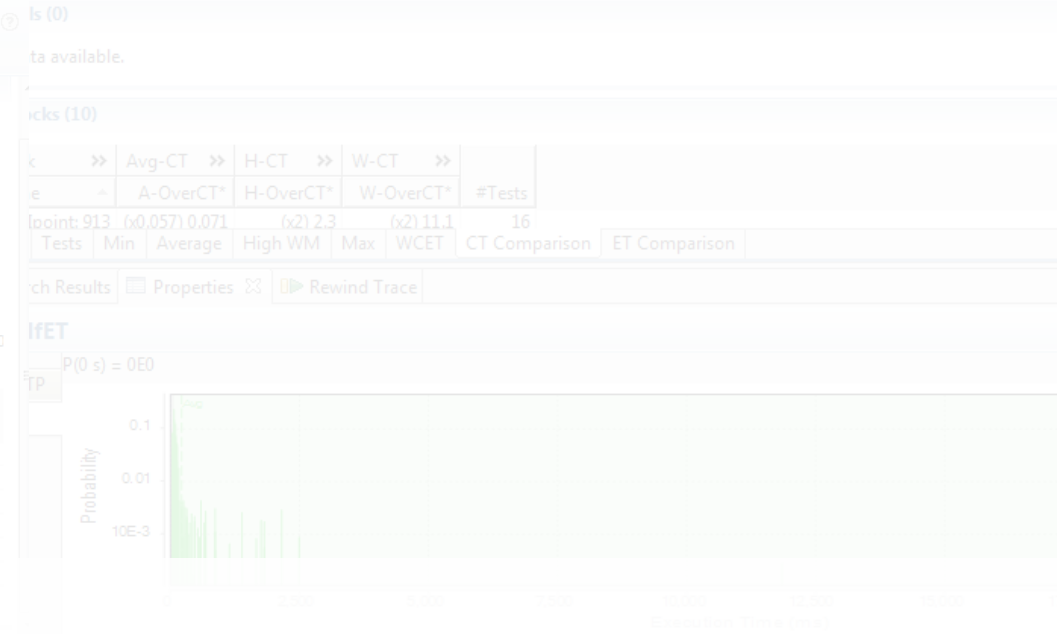
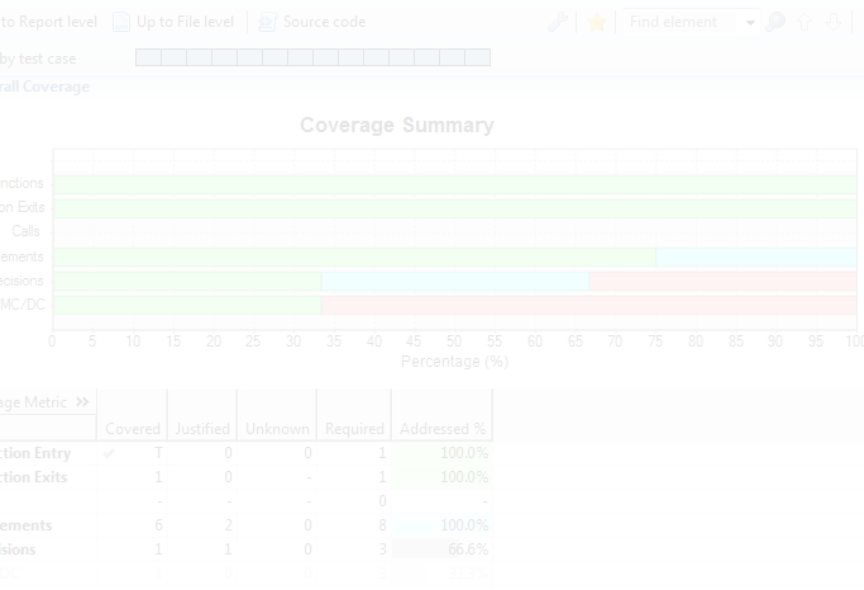
- Successful test flight for an engine running RVS-instrumented code in-flight (Ada)
- Launch of RVS 3.5 and updated DO-178C qualification kit next week
- New development: RapiTest Framework – test driver tool
- DO-178C training day in London (30th June)
- Training for the York 100 km bike ride



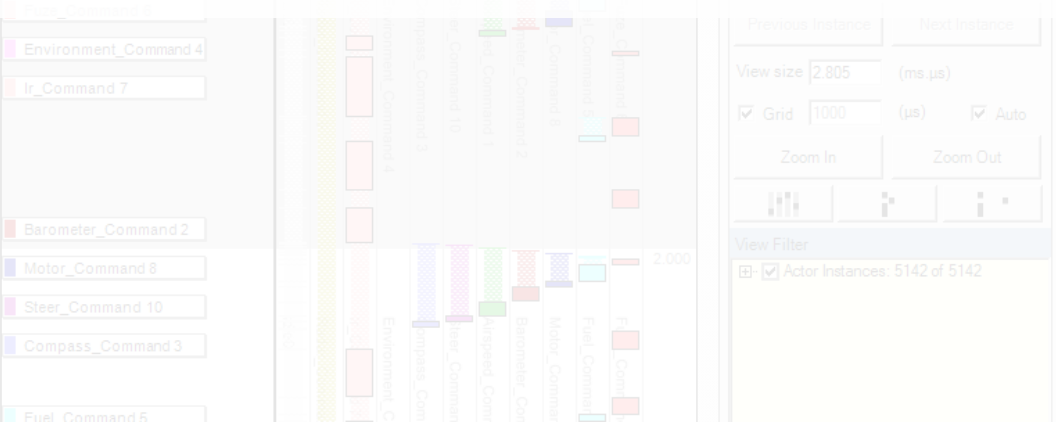
DO-178C Colloquium

 30th June 2016

[Find out more »](#)



Rapita Verification Suite



```

40  begin;
41  T_valid := New_Stage_Timer(Current_Stage);
42  if T_valid < 0 then
43    Clock_Reset_T := T_valid;
44    valid := T_valid;
45  else
46    valid := T_valid;
47  end if;
48  end;
49  end Update_Stage;
50  end if;
51  end Update_Stage;
52  end if;
53  end;
54  end;
55  end;
56  end Update_Stage;
57  end;
58  end;
59  end;
60  end;
61  end;
62  end;
63  end;
64  end;
65  end;
66  end Update_Stage;
67  end;
68  end;
69  ----- Public subprograms
70  procedure Set_stage(New_stage : in Stage)
71  --# global out current_stage, stage_timer;
72  --# in out r_clock_time;
73  begin
74  end;
75  end Set_stage;
76  end;
77  end;
78  end;
79  end;
80  end;
81  end;
82  end;
83  end;
84  end;
85  end;
86  end;
87  end;
88  end;
89  end;
90  end;
91  end;
92  end;
93  end;
94  end;
95  end;
96  end;
97  end;
98  end;
99  end;
100 end;

```

Search Results | Properties | Rewind Trace

destruct.adb:51
51 if T_valid and then
Source 52

Function: destruct...OV_BRANCHES, COV_CALLS, COV_FUNCTIONS, COV_MCDC, COV_FUNCTION_EXITS

Filter by test case

Overall Coverage

Coverage Summary



RapiCover

Your *solution* for structural code coverage analysis

Coverage Metric	1	0	0	3	Percentage
Function Entry	1	0	0	1	100.0%
Function Exits	1	0	0	1	100.0%
Calls	1	0	0	1	100.0%
Statements	6	2	0	8	100.0%
Decisions	1	1	0	3	66.6%
MC/DC	1	0	0	3	33.3%

Static | Coverage | Functions | Calls | Statements | Decisions | MC/DC | Justifications

Search Results | Properties | Rewind Trace

Coverage Justifications

- RVS 3.4 allows coverage gaps to be addressed by justifications.
- Intelligently identifies code changes that require review of previous justifications.

The infographic is divided into several sections. At the top, a purple banner states "It's difficult to get 100% code coverage". Below this, four categories of coverage gaps are listed: "Missing requirements" (magnifying glass icon), "Missing test cases" (gears icon), "Dead code" (graveyard icon), and "Deactivated code" (toggle switch icon). Underneath, three reasons for these gaps are shown: "Defensive programming" (shield icon), "Impossible events" (triangle icon), and "Compiler errors" (circular arrows icon). A black bar asks "What can I do about it?". Below this, four action buttons are provided: "Remove code", "Add / Fix requirements", "Justify why code won't be executed", and "Justify why it isn't possible to test". The next section is a large orange circle containing a flowchart: "Identify coverage hole" (person icon) leads to "Write justification" (two people icon), which leads to "Change source code" (person and code icon). This leads to a central circle with "Manually review / retest all justifications" (group of people icon). From this center, arrows point to "Build" and "Re run tests". The bottom section is green and says "Save time and effort by migrating justifications between builds", accompanied by clock icons and a group of people icon. The footer contains contact information: "For any questions about RVS, email: enquiries@rapitasystems.com". Logos for RAPITA and other partners are at the very bottom.

Incremental Coverage

- Conduct structural coverage analysis quicker by allowing RVS to cleverly optimize the partitioning of your instrumentation between test runs.

The infographic is a vertical flowchart illustrating the benefits of incremental coverage analysis. It starts with a header showing four gauges for Memory and Execution time. A question box asks, "What's the problem with this approach?" followed by an answer box stating, "Manually splitting instrumentation into several, partially-instrumented builds is difficult & time-consuming." Below this, four possible combinations of instrumentation are shown with gauges containing question marks. A second question box asks, "How can RVS help?" This is followed by three boxes: "Use less memory per instrumentation point", "Use RVS' incremental coverage feature", and "Use fewer instrumentation points". Below these are three gauges for Memory and Execution time. The final section shows three runs: Run 1 (Fully instrumented source), Run 2 (Remove instrumentation executed in run 1), and Run 3 (Remove instrumentation executed in run 2). A diagram shows source code with red and green highlights indicating instrumentation changes between runs. A key at the bottom explains the colors: Uninstrumented source (blue), Instrumented source (not executed) (red), and Instrumented source (executed) (green).

KEY: Source code Instrumentation

Q. "What's the problem with this approach?"

A. "Manually splitting instrumentation into several, partially-instrumented builds is difficult & time-consuming."

Possible combination 1 Possible combination 2 Possible combination 3 Possible combination 4

Q. "How can RVS help?"

Use less memory per instrumentation point Use RVS' incremental coverage feature Use fewer instrumentation points

Run 1 Fully instrumented source Run some tests. Run 2 Remove instrumentation executed in run 1. Run some more tests. Run 3 Remove instrumentation executed in run 2. Run remaining tests.

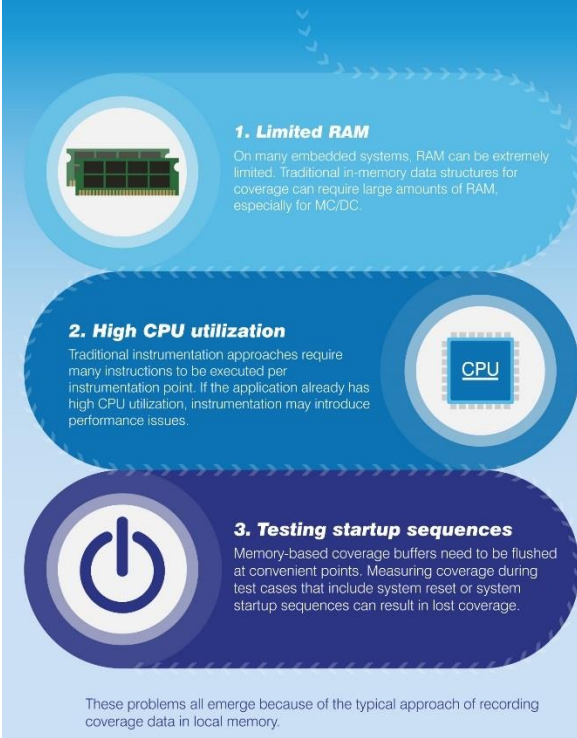
KEY: Uninstrumented source Instrumented source (not executed) Instrumented source (executed)

For any questions about RVS, email:

Livemaps

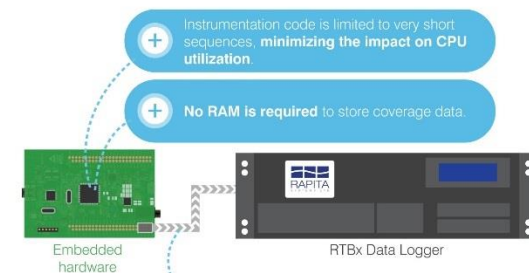
- Efficient code coverage supporting reboot, reset, error conditions, fault tolerance

What are the problems with structural coverage analysis on embedded systems?




How can this be addressed?

As an alternative, it is possible to send coverage information directly from target interfaces, such as digital I/O ports, debug interfaces or external memory buses. Coverage data is then collected via external devices, such as data loggers.



Language Ada Features/Enhancements

- Ada: Support for GNAT Pro 7.3 and 7.4
 - Ada: Improved support for various Ada language features (case statements, renames, dispatching subprograms etc)
 - Ada: Stubbing of closures for unit-test
 - Ada: Flexible unit-test of Ada subprograms with stubbing
 - Ada: 2012 support – looking good, but not complete
-
- C: “Duff’s Device” support!




```
switch( pOp->opcode ){
...
case OP_SorterNext: { /* jump */
    VdbeCursor *pC;
    int res;

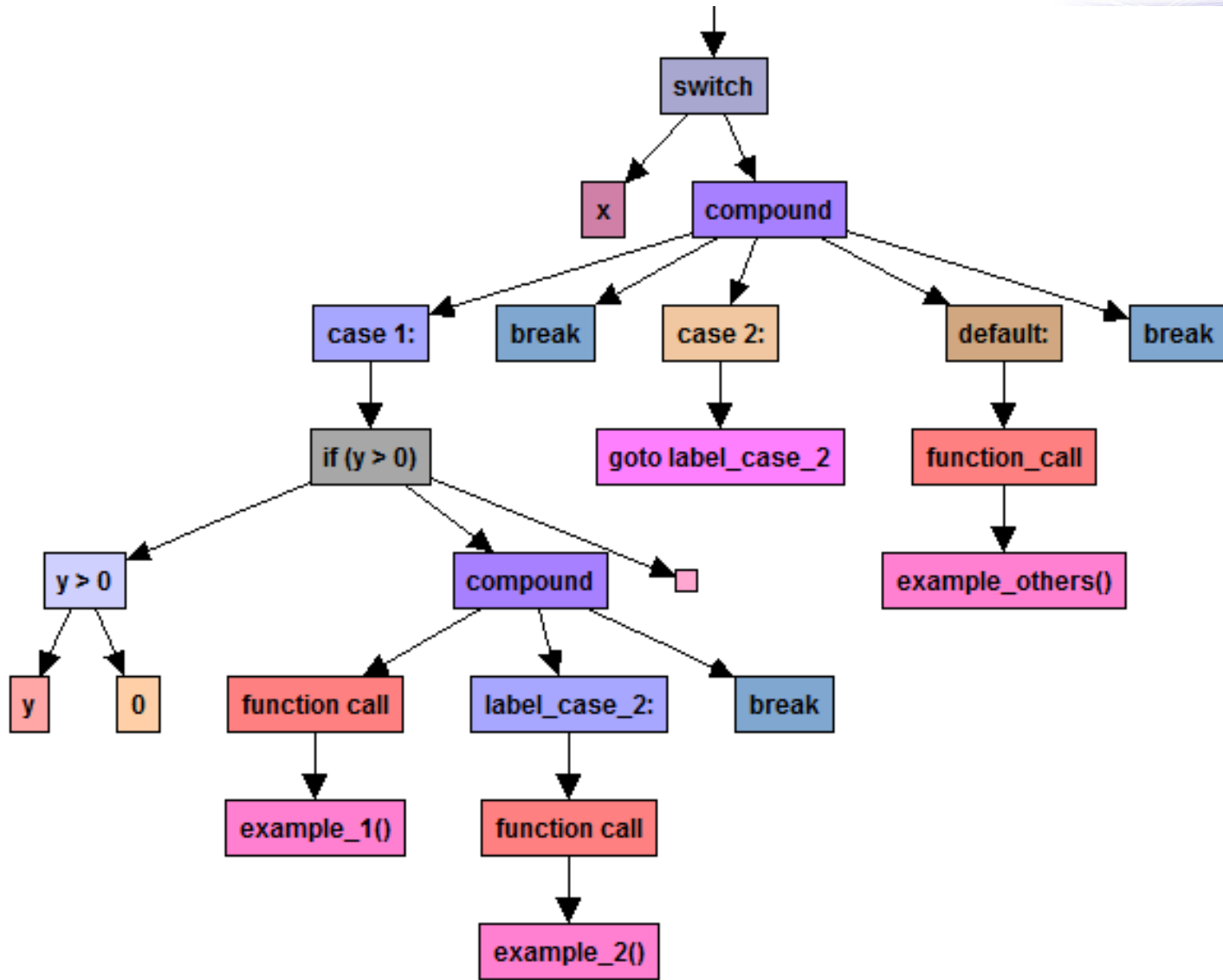
    pC = p->apCsr[pOp->p1];
    ...
    res = 0;
    ...
    goto next_tail;
case OP_PrevIfOpen:      /* jump */
case OP_NextIfOpen:     /* jump */
    if( p->apCsr[pOp->p1]==0 ) break;
    /* Fall through */
case OP_Prev:            /* jump */
case OP_Next:           /* jump */
    ...
    pC = p->apCsr[pOp->p1];
    res = pOp->p3;
```

Duff's Device

```
int n = (count + 7) / 8;
    switch (count % 8) {
case 0: do { *to = *from++;
case 7:      *to = *from++;
case 6:      *to = *from++;
case 5:      *to = *from++;
case 4:      *to = *from++;
case 3:      *to = *from++;
case 2:      *to = *from++;
case 1:      *to = *from++;
            } while (--n > 0);
    }
```



```
switch (x) {
  case 1:  if (y > 0) {
            example_1 ();
          }
  case 2:  example_2 (); break;
          }
  default: example_others (); break;
}
```



Overview of products & services

Services

Target Integration Service

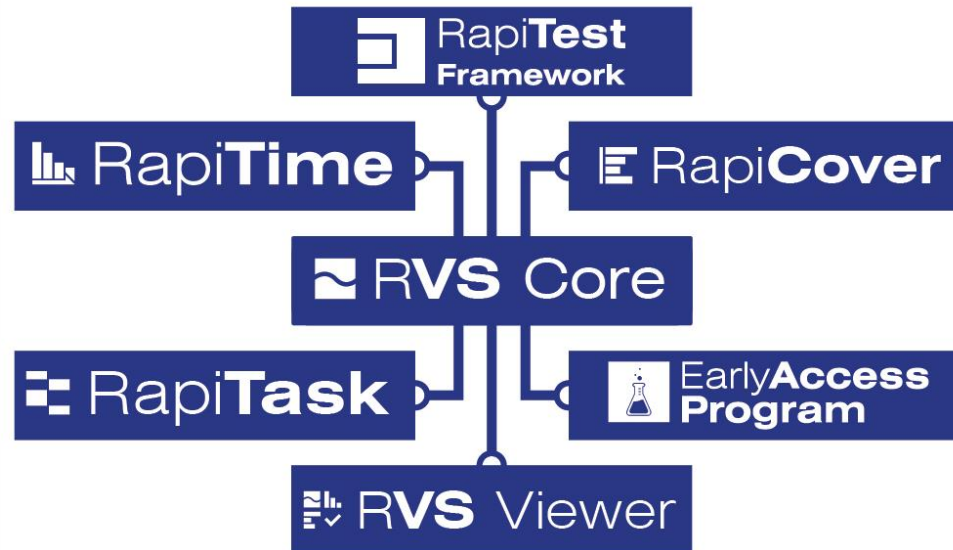
Qualification pack DO-178B/C, ISO 26262

Training

Engineering Services

Support

Software

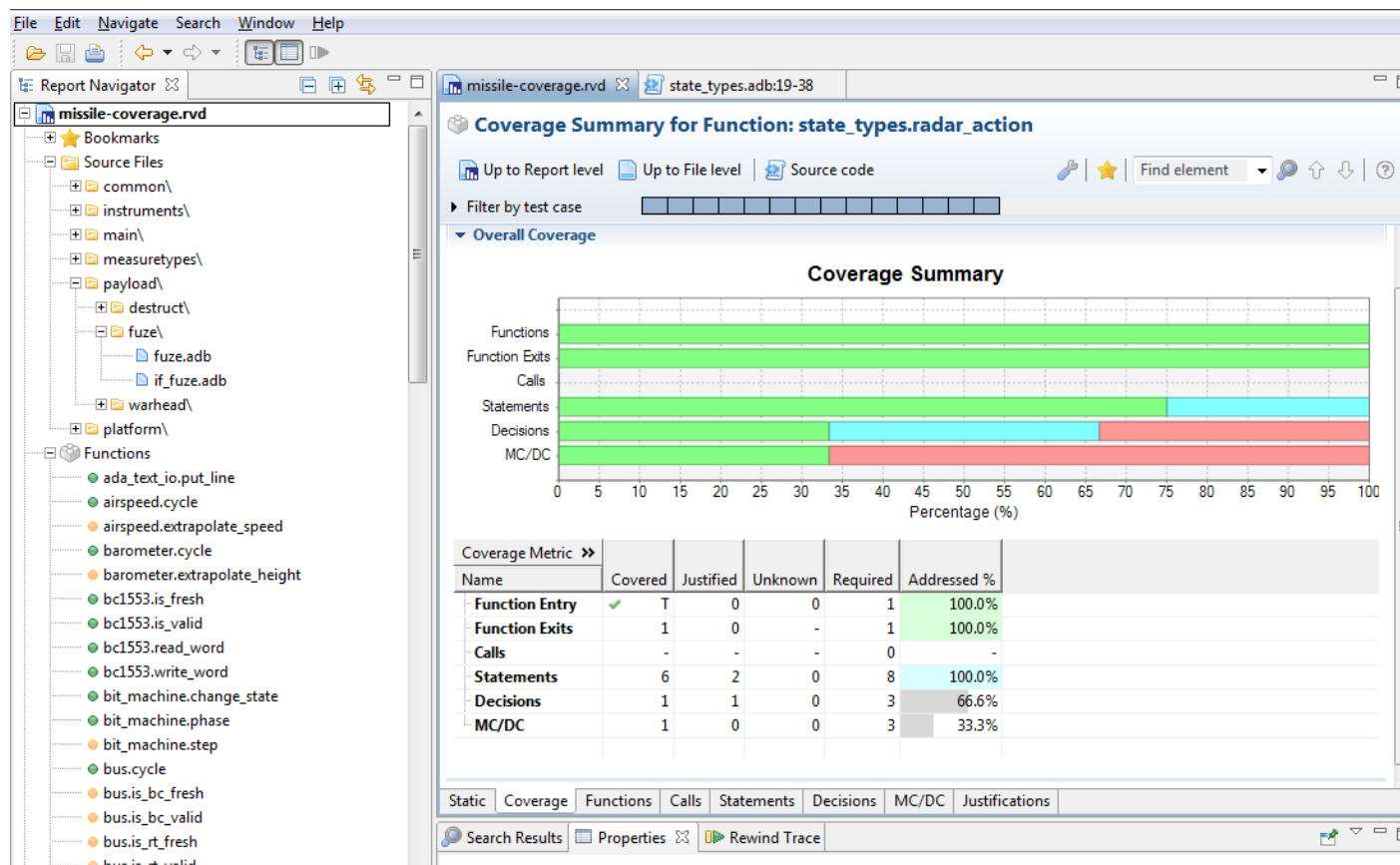


Data logger

RTBx

What is RapiCover?

- An advanced structural code coverage analysis tool designed specifically to work with embedded targets



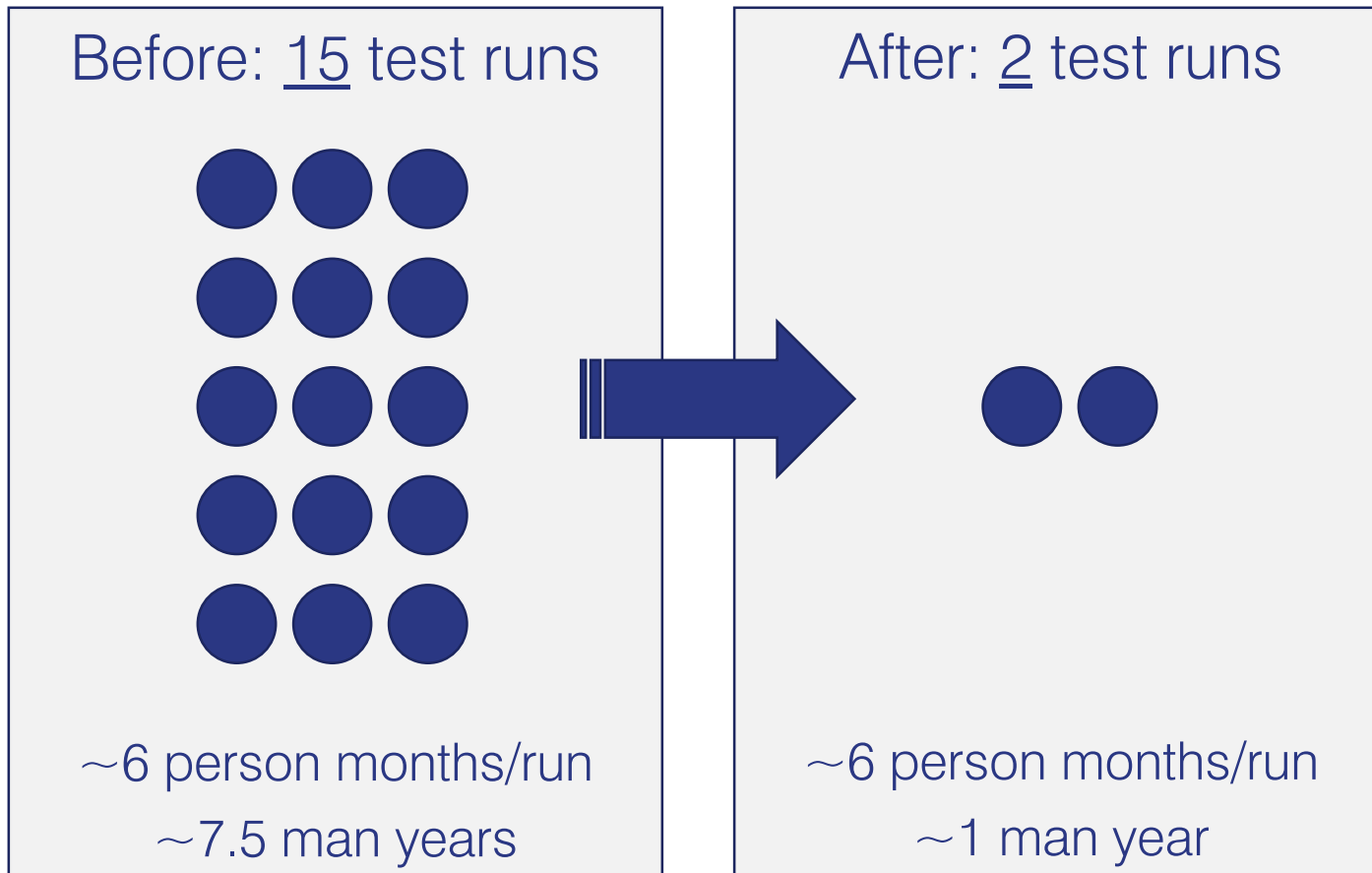
Advantages / benefits of using RapiCover

- **Complete coverage testing in fewer test cycles**
 - Best-in-class on-target overheads: minimal memory footprint / performance impact
- **Designed to be adapted to work with your system,** rather than imposing a rigid approach
- **Reduce reporting effort**
 - Combine multiple reports
 - Justify untested code
- **Reduce certification risk**
 - RapiCover handles many complex coding structures not supported by other code coverage tools



Case study

- One customer reduced the number of test runs from 15 to just 2 thanks to RapiCover's low overheads





on >>	Avg-CT >>	H-CT >>	W-CT <<				#Tests
▲	A-SelfCT*	H-SelfCT*	W-Freq	W-SelfET	W-SelfCT	W-SelfCT%	
it4_array	(x0.057) 11.1	(x2) 733.8	2	20,089	40,177.4	1.5%	16

(0)

available.

ks (10)

RapiTime

Timing verification and optimization

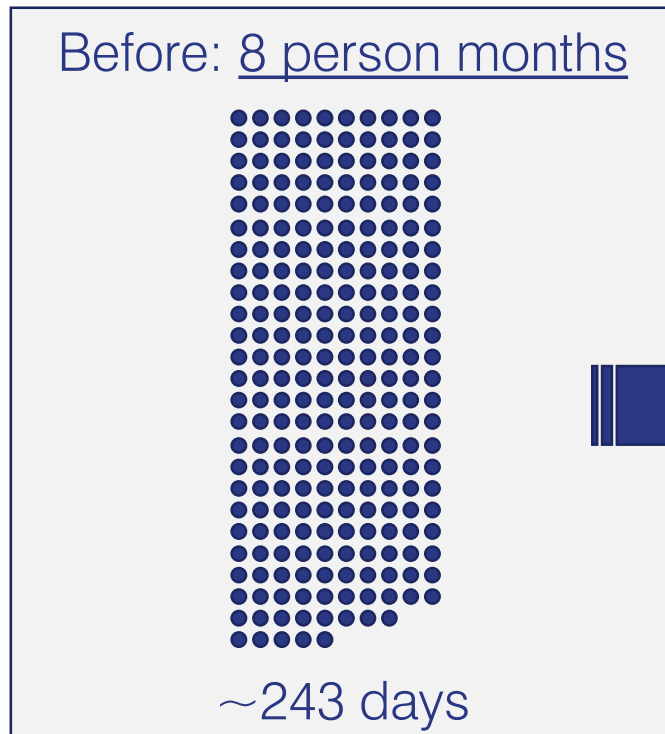
P(0 s) = 0E0



Case study

- One customer, who's manual timing analysis process took 8 months, managed to reproduce the same results in 1 day with RapiTime

Customer's manual process



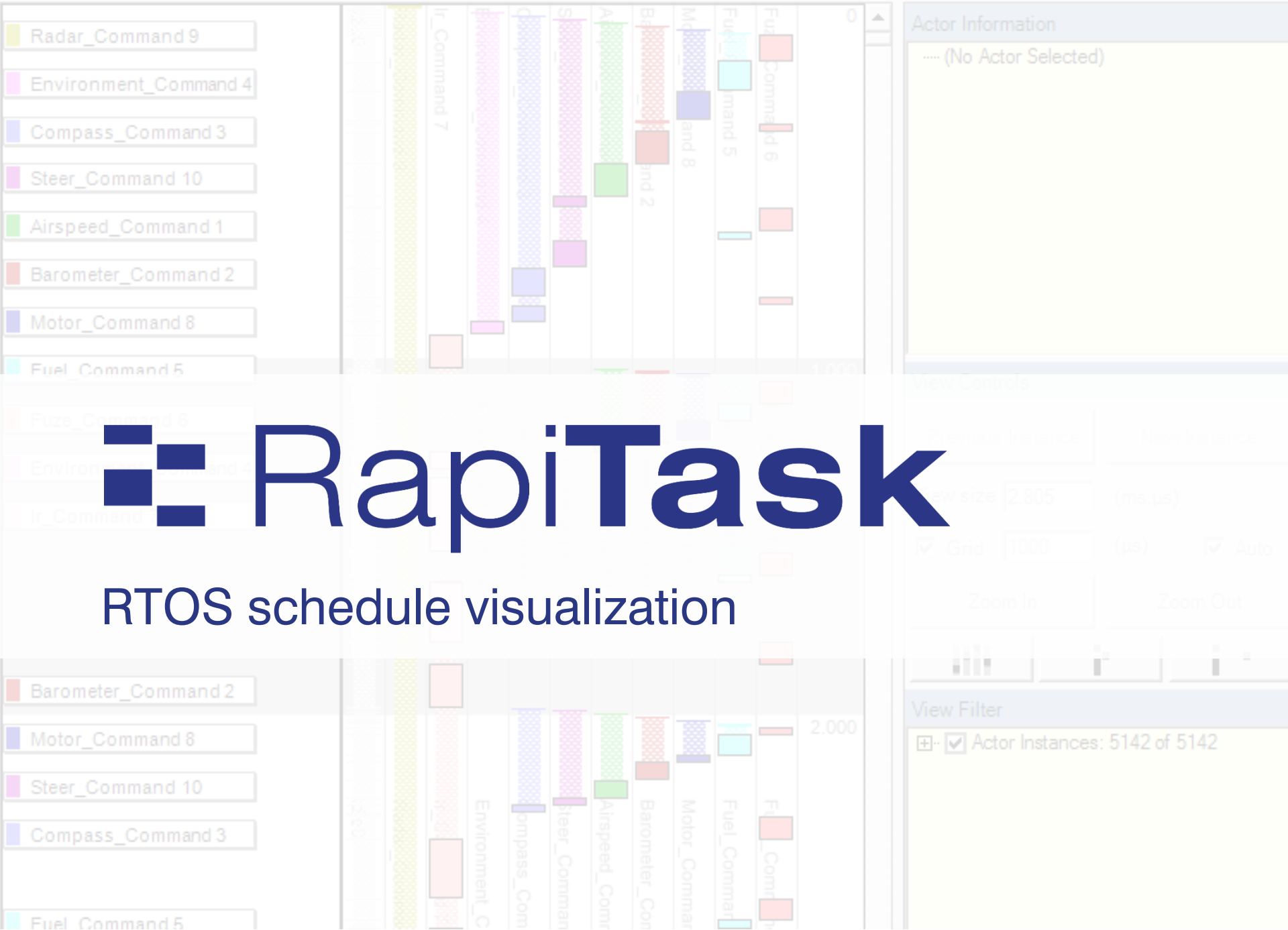
 **RapiTime**





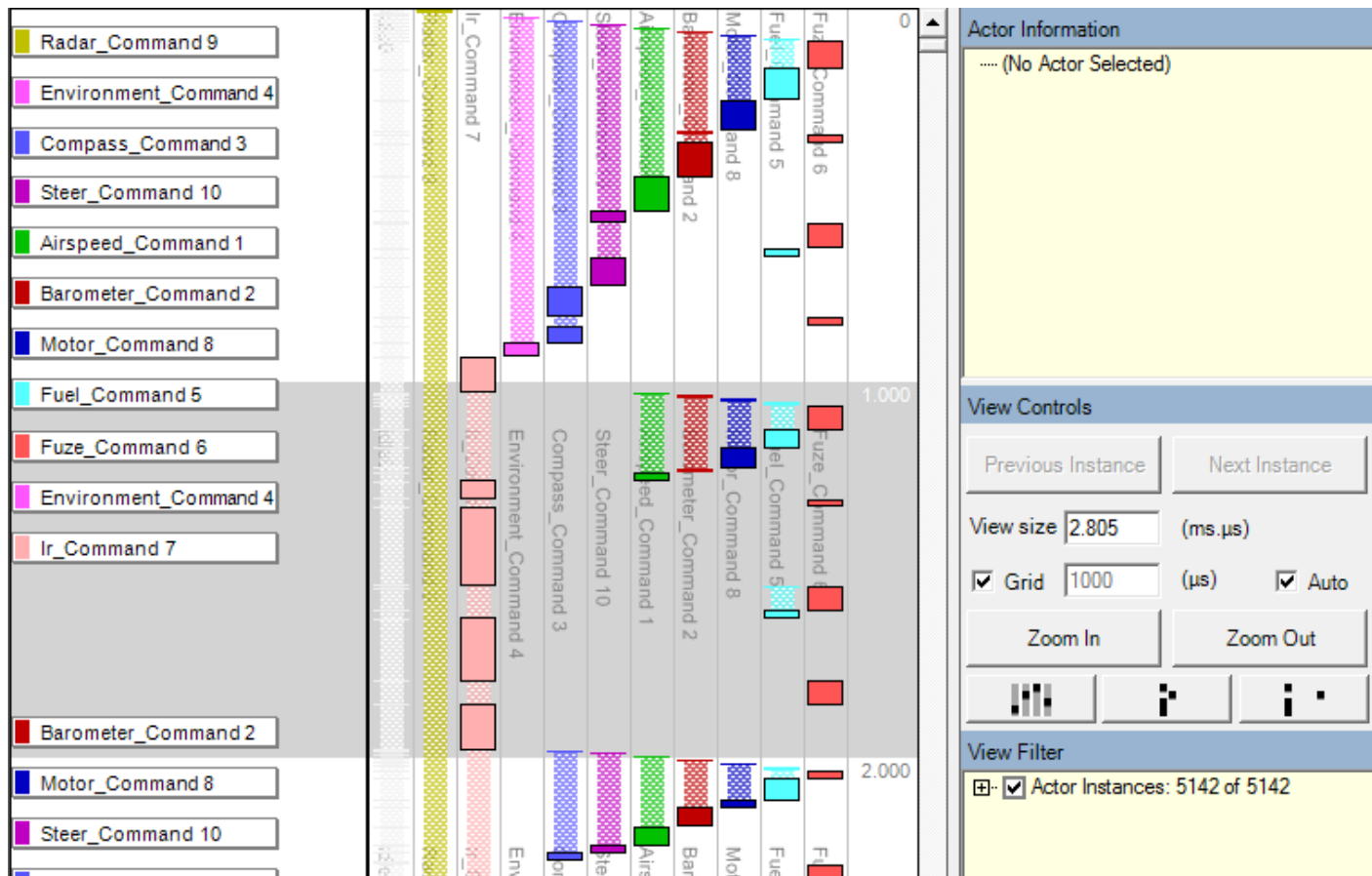
RapiTask

RTOS schedule visualization



What is RapiTask?

- A tool which provides visualisation of RTOS scheduling and event tracing for complex embedded systems



Next steps

Find out more on our website:

RVS »

RapiCover »

RapiTime »

RapiTask »

RTBx »

RapiTestFramework »

Questions?

Please email enquiries@rapitasystems.com
or complete our online enquiry form:

Enquire online »