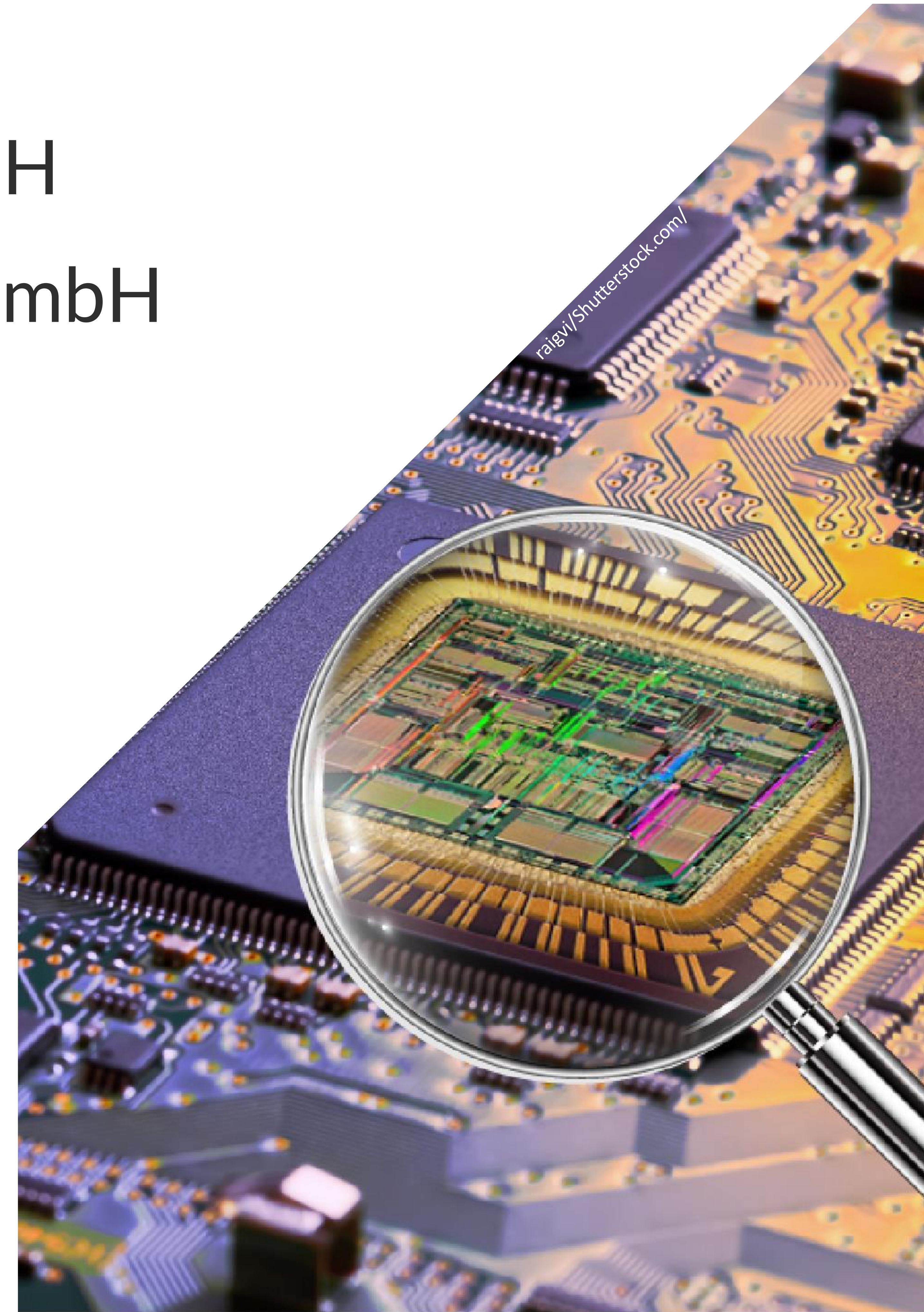
Structural High-level Tests

Alexander Weiss, Accemic Technologies GmbH

Martin Heininger, HEICON Global Engineering GmbH



ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Motivation

Bug-free Software is a Myth.

Trends:

- More complex integrated systems
- Software-driven safety-critical applications (e.g. autonomous driving)

➢ Increased number of critical software defects

Software defects are a considerable cause for

- Project schedule overruns (~60%) [1]
- Project budget overruns (~50%) [1]
- Project cancellation (~10-20%) [2,3]
- Higher maintenance costs
  (Airbus study[4] : Cost of a non-frequent failure: up to 500,000 €)
- Injuries, accidents, recalls


Toyota "Unintended Acceleration" Has Killed 89

A 2005 Toyota Prius, which was in an accident, is seen at a police station in Harrison, New York, Wednesday, March 10, 2010. The driver of the Toyota Prius told police that the car accelerated on its own, then lurched down a driveway, across a road and into a stone wall. (AP Photo/Seth Wenig) / AP PHOTO/SETH WENIG

Unintended acceleration in Toyota vehicles may have been involved in the deaths of 89 people over the past decade, upgrading the number of deaths possibly linked to the massive recalls, the government said Tuesday.

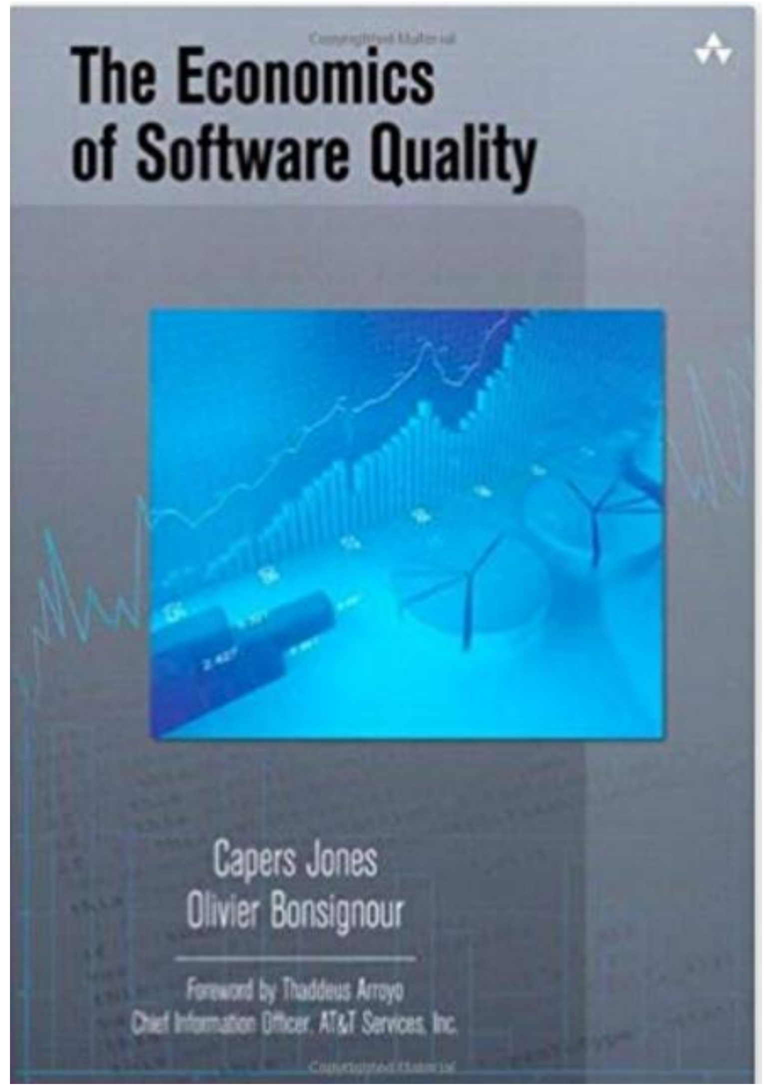
The National Highway Traffic Safety Administration said that from 2000 to mid-May, it had received more than 6,200 complaints involving sudden acceleration in Toyota vehicles. The reports include 89 deaths and 57 injuries over the same period. Previously, 52 deaths had been suspected of being connected to the problem. http://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/

[1] Gartner Research
[2] Emam et al, "A Replicated Survey of IT Software Project Failures."
[3] Standish Chaos Report
[4] Hanke et al, "Assessment of multi-core integration infrastructure" Munich, 2014.

Motivation

Bug-free Software is a Myth.

McKinsey & Company (2018):
"**Snowballing complexity** is causing
significant software-related quality issues …"



Size of SW projects in FP	Average Defect Potential (Defects / FP)	Defect Removal Efficiency
100	3,00	98%
10.000	6,25	96%
1.000.000	8,25	86%

1 MLOC (C-Code) => 6.250 FP => ~35.000 defects => ~ 2.500 Post Release Defects

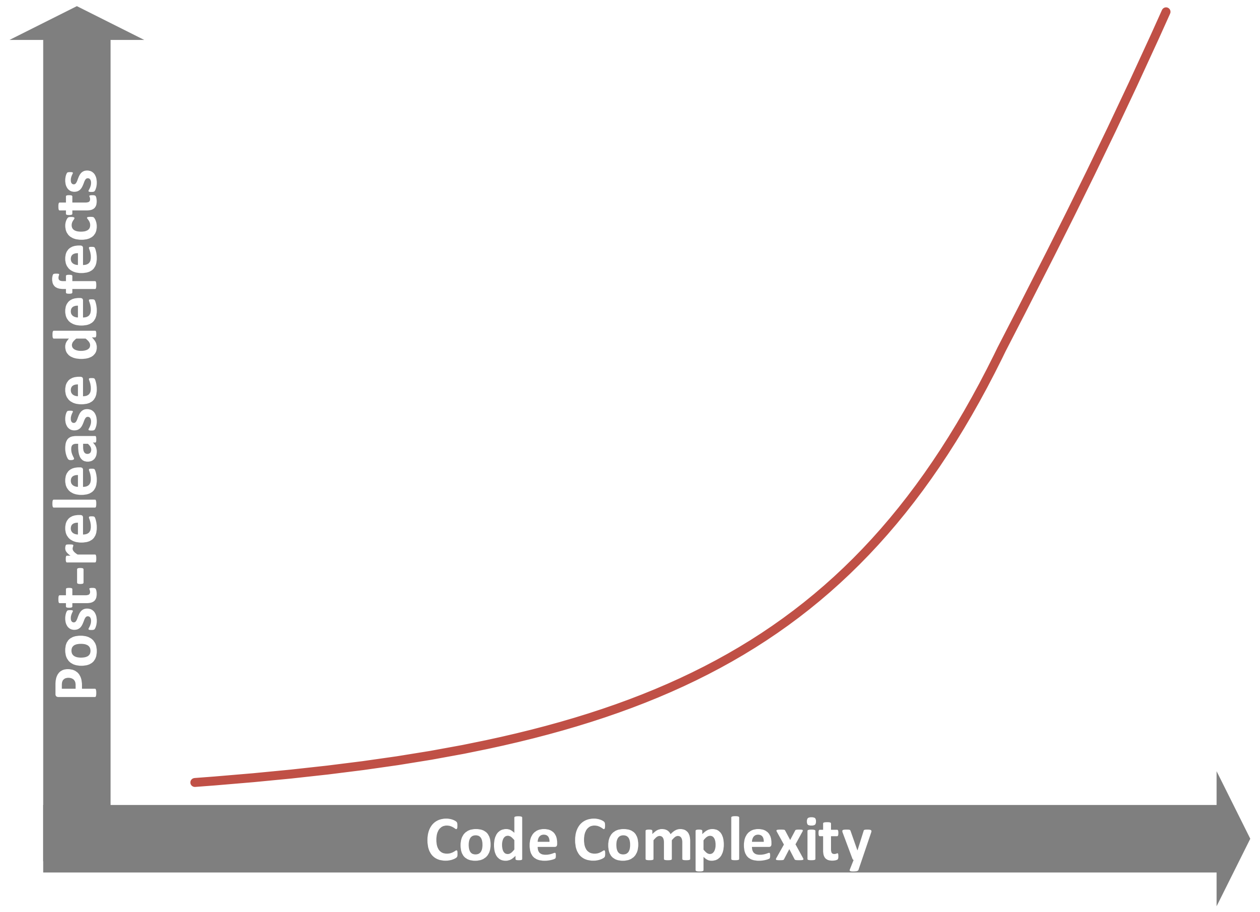
* 1 FP (Function Point): ~160 LOC (C language), ~64 LOC (ADA), ~32 LOC (C++)

Motivation

Bug-free Software is a Myth.

Increasing size of SW projects ✖ [Average Defect Potential increases with the size of SW projects. ÷ Defect Removal Efficiency decreases with the size of SW projects.]

✚ [Additional non-deterministic defects, caused by multicore interferences]



=

- Challenges for embedded software architecture
- Challenges for testing
- Challenges for maintenance

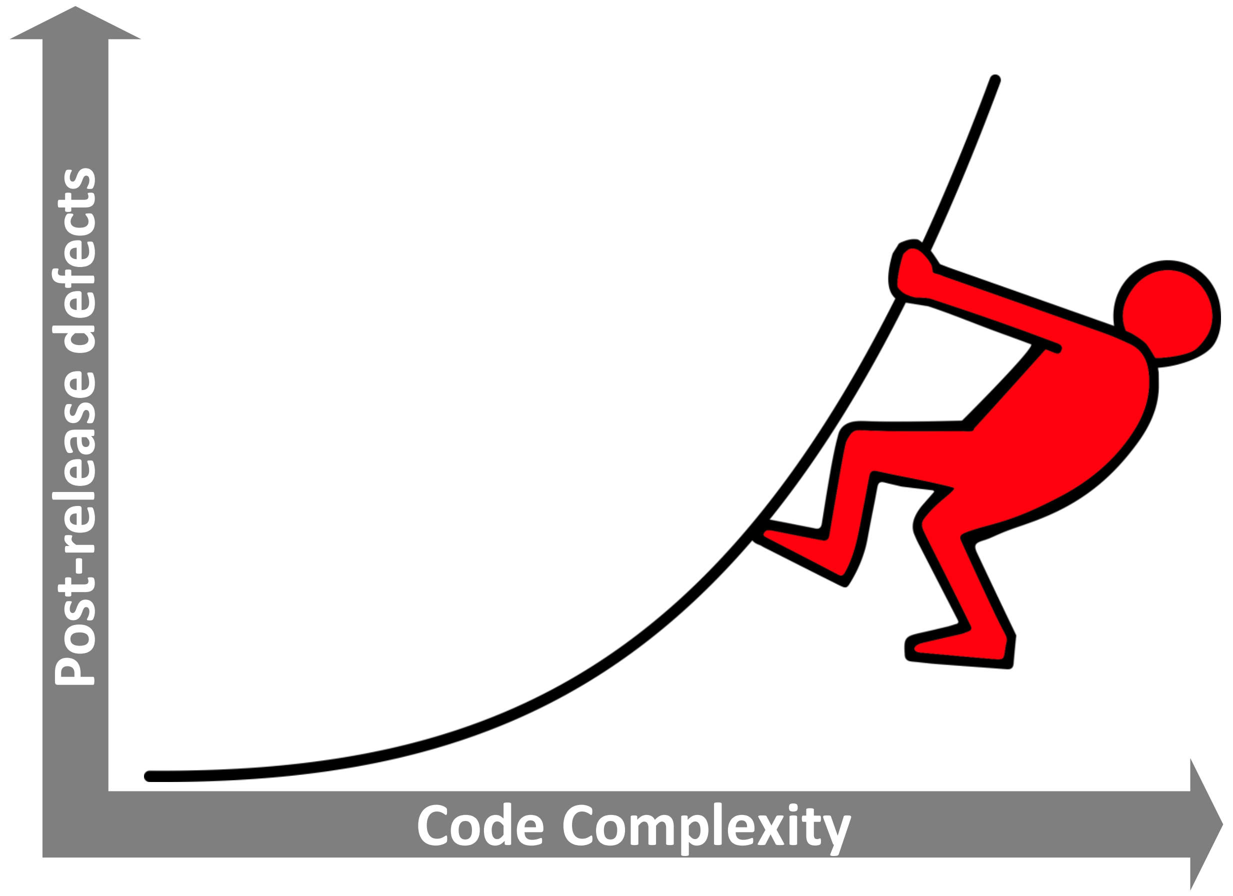
What we can do

Challenges:

- Improve development process
- Improve test effectiveness and test efficiency
- Improve in-field analysis and debug capabilities
- ….

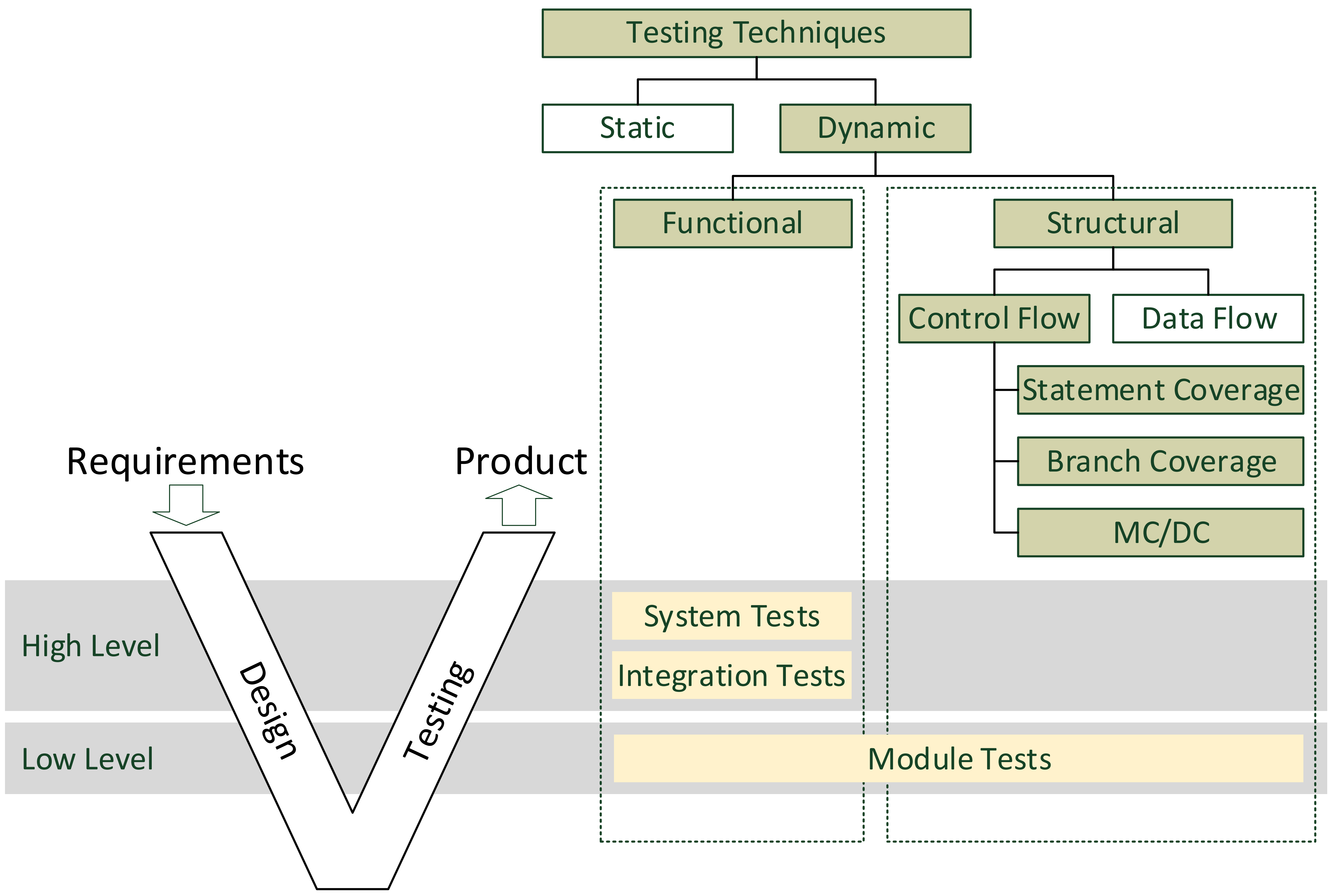
Our contribution:

- Regain observation capabilities in highly integrated multicore systems
- Enable continuous, non-intrusive online observation and automated validation
  - ➢ Structural coverage at higher test levels
  - ➢ Timing analysis at higher test levels



Dynamic Tests

Overview



Functional tests (black box)

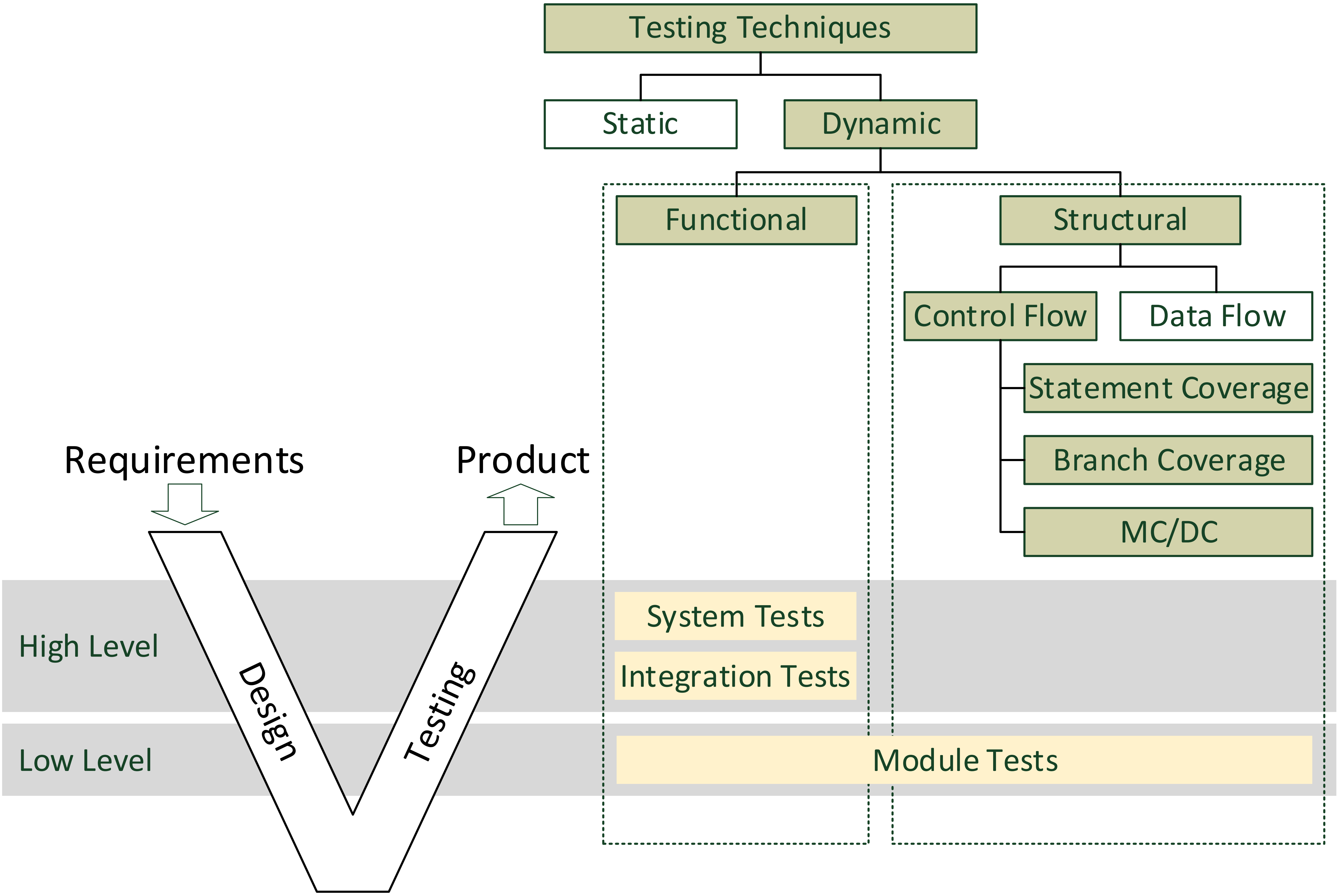
Test the implementation of the functional requirements.

Structural tests (white box)

Did my functional tests use all my code?

Dynamic Tests

Overview



Functional tests

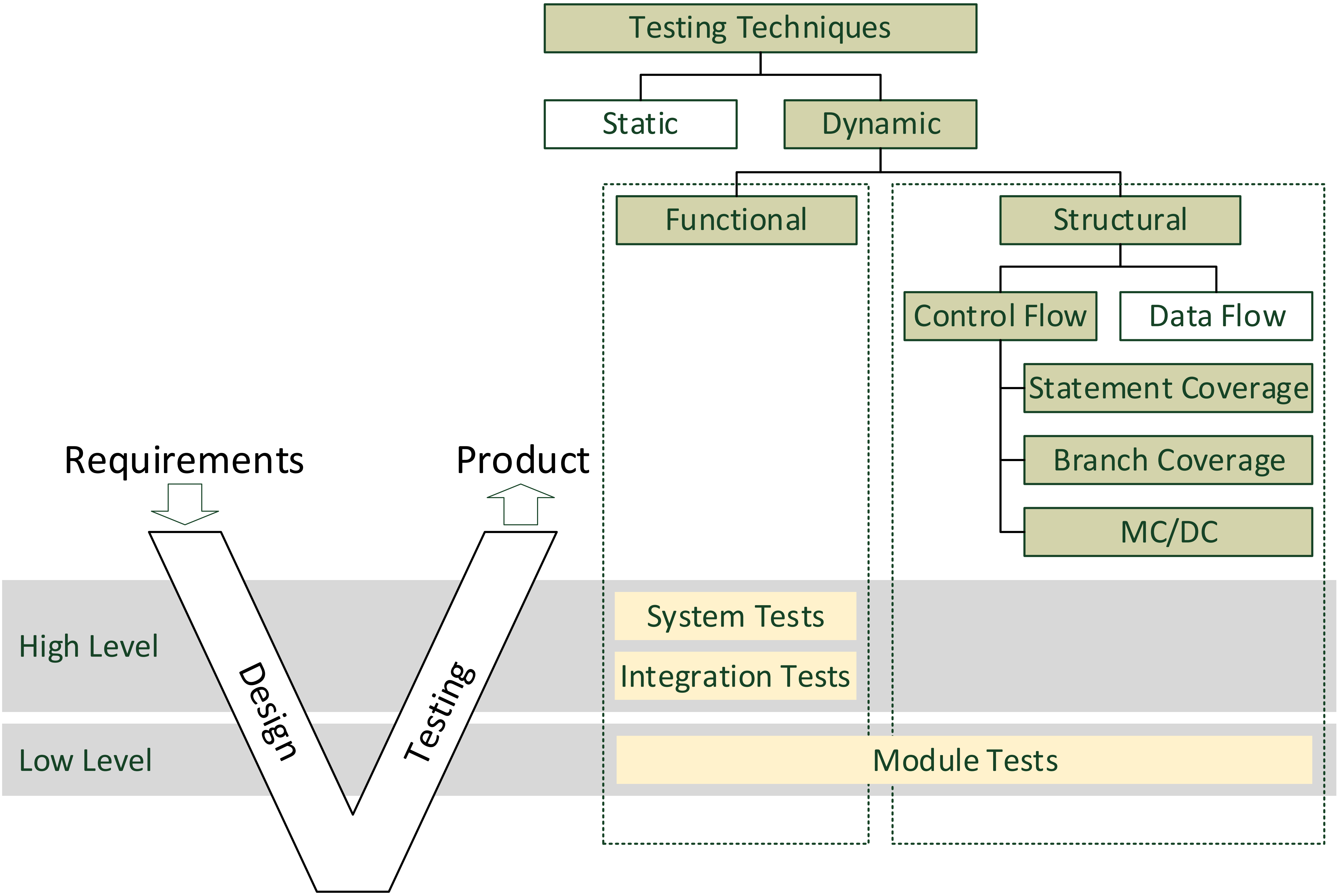
Test the implementation of the functional requirements.

Structural tests

Did my functional tests use all my code?

Dynamic Tests

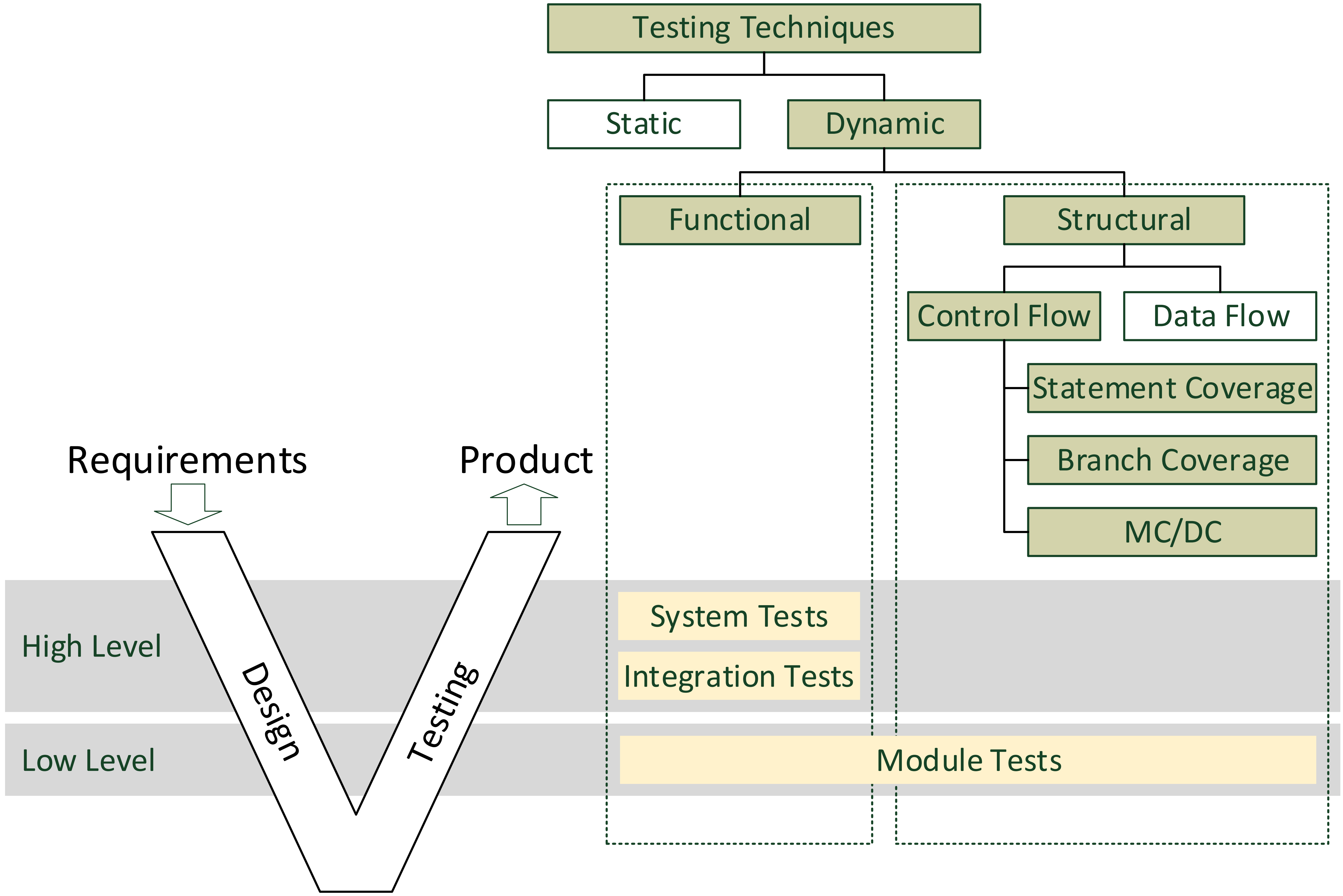
Overview



$$\textit{Statement Coverage} = \frac{\#\ \textit{of Executed Statements}}{\#\ \textit{of Statements}}$$

Dynamic Tests

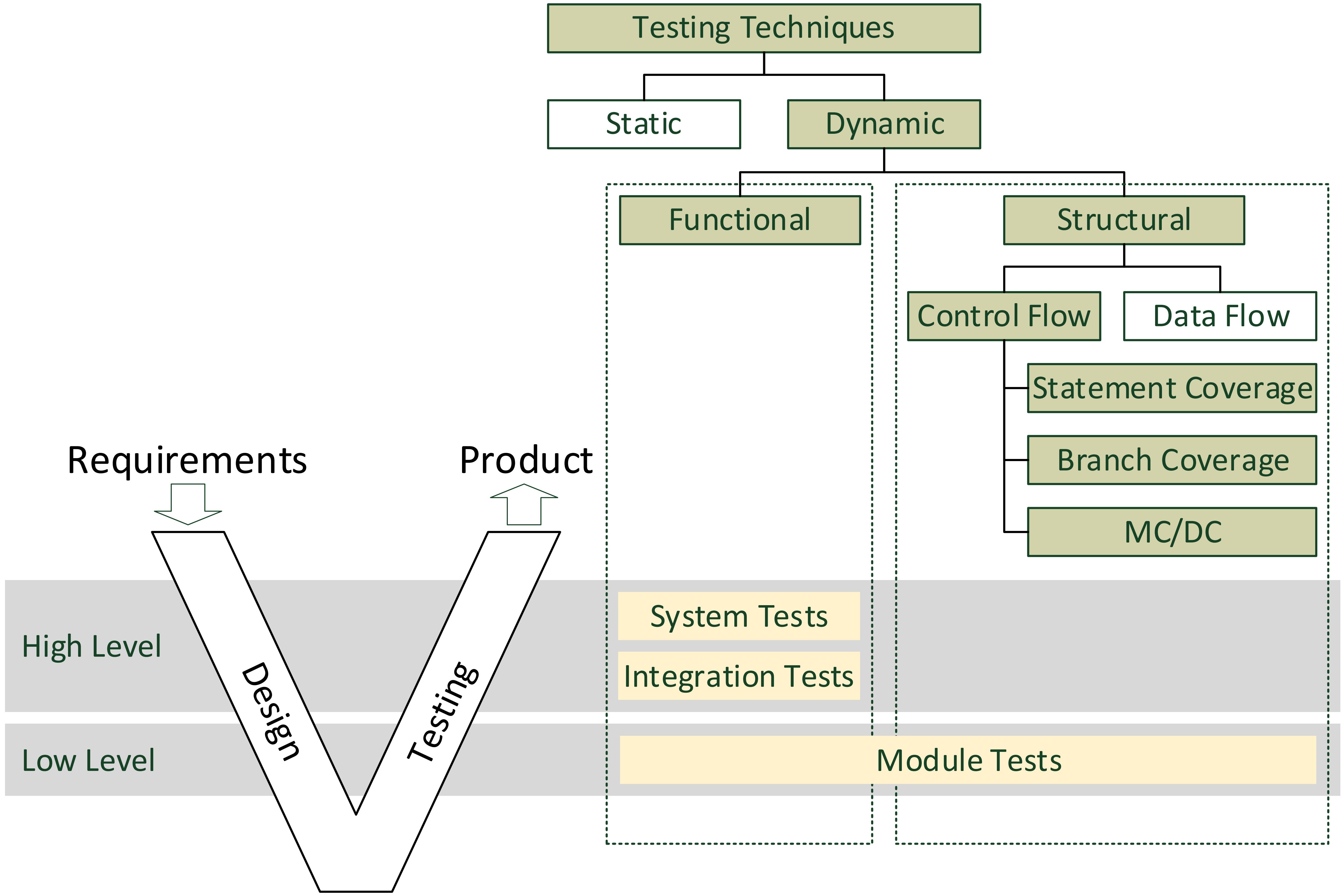
Overview



$$Branch\ Coverage = \frac{\#\ of\ Executed\ Branches}{\#\ of\ Branches}$$

Dynamic Tests

Overview



A Practical Tutorial on Modified Condition/ Decision Coverage

MC/DC

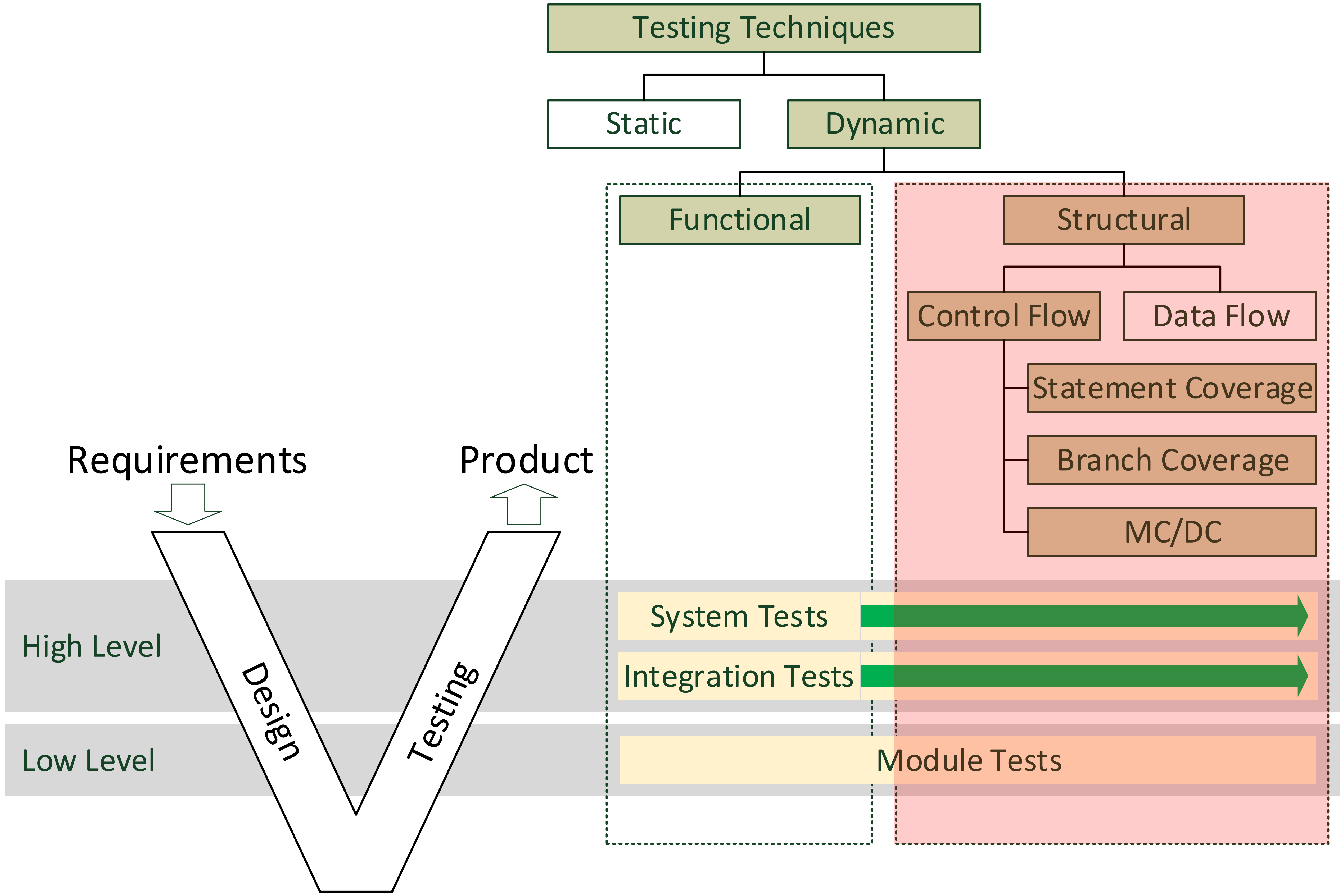
Each condition in a decision has been shown to independently affect that decision's outcome.

A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010057789.pdf

Dynamic Tests

Overview



Functional tests (black box)

Test the implementation of the functional requirements.

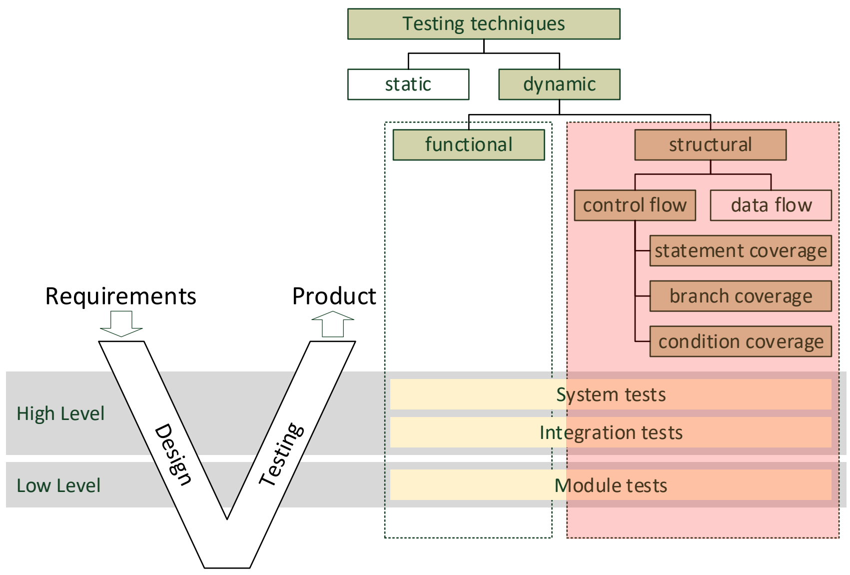
Structural tests (white box)

Did my functional tests use all my code?

IMPROVEMENT:

Structural tests for high-level tests.

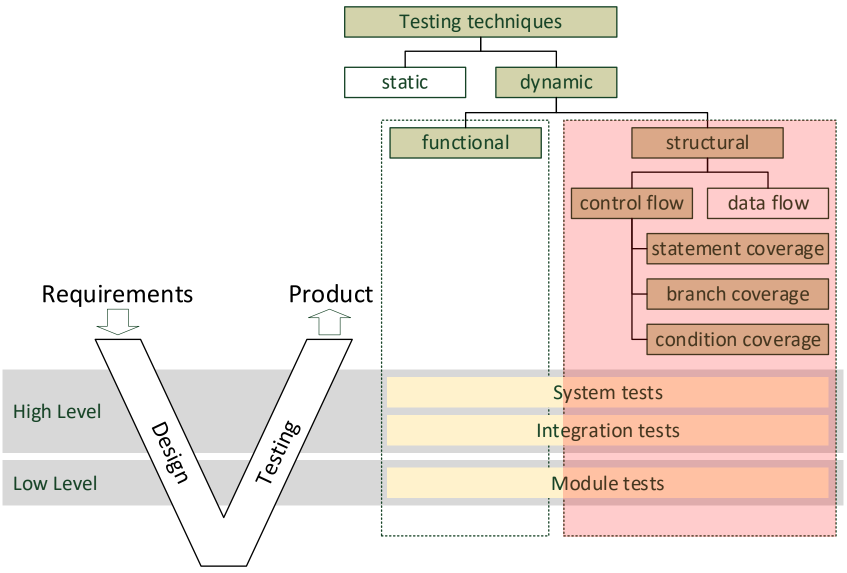
High-level Structural Tests



Requirements für structural tests

	Long observation	No change of timing behavior	No change of memory footprint	
System tests	Y	Y	Y	Limitations by using SW instrumentation
Integration tests	Y	Y	Y	Limitations by using SW instrumentation
Module tests	N	N	N	SW instrumentation is perfect.

High-level Structural Tests



Requirements für structural tests

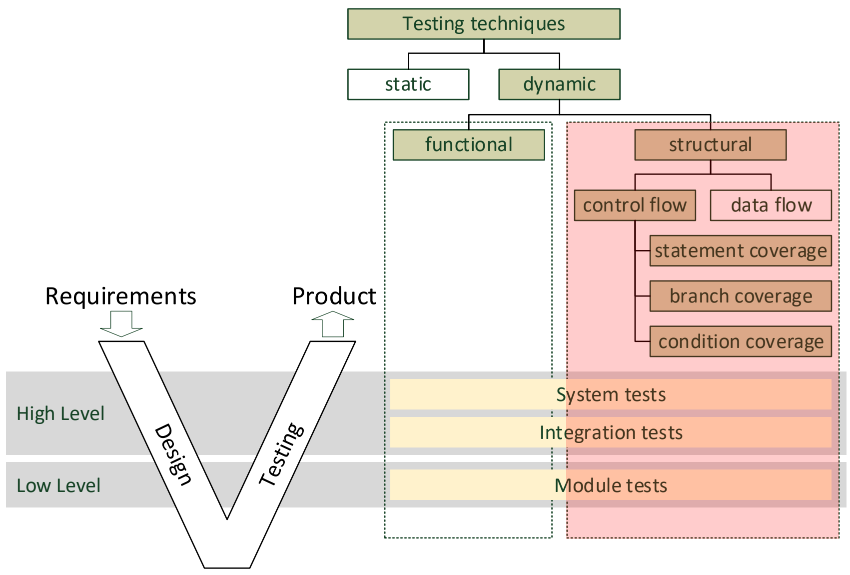
	Long observation	No change of timing behavior	No change of memory footprint	
System tests	Y	Y	Y	Limitations by using SW instrumentation
Integration tests	Y	Y	Y	Limitations by using SW instrumentation
Module tests	N	N	N	SW instrumentation is perfect.

State of the art:
Measuring code coverage using SW instrumentation

ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests



Requirements für structural tests

	Long observation	No change of timing behavior	No change of memory footprint	
System tests	Y	Y	Y	Limitations by using SW instrumentation
Integration tests	Y	Y	Y	Limitations by using SW instrumentation
Module tests	N	N	N	SW instrumentation is perfect.

New approach:
Non-intrusive and continuous observation

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Non-intrusive and Continous Observation

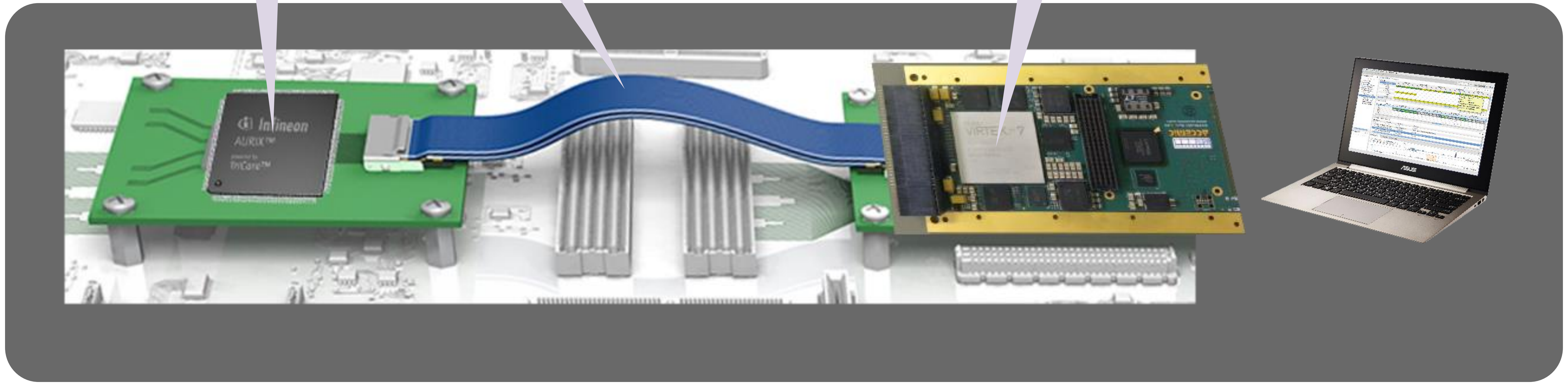
Processor
(Aurix, Cortex-A, Intel Atom, QorIQ)

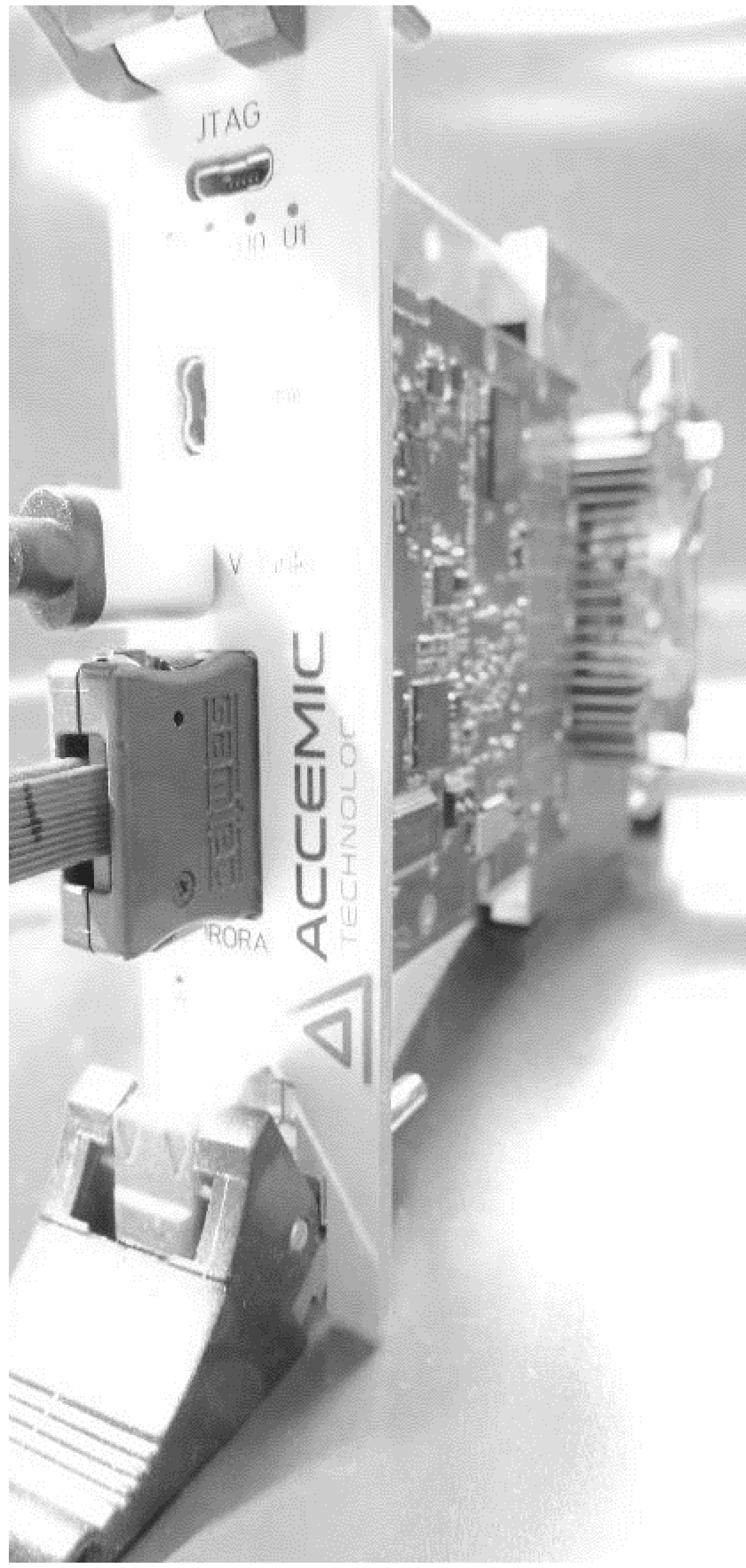
Multi Gbps Synchronization

CEDARTOOLS

Live Synchronized Digital Twin

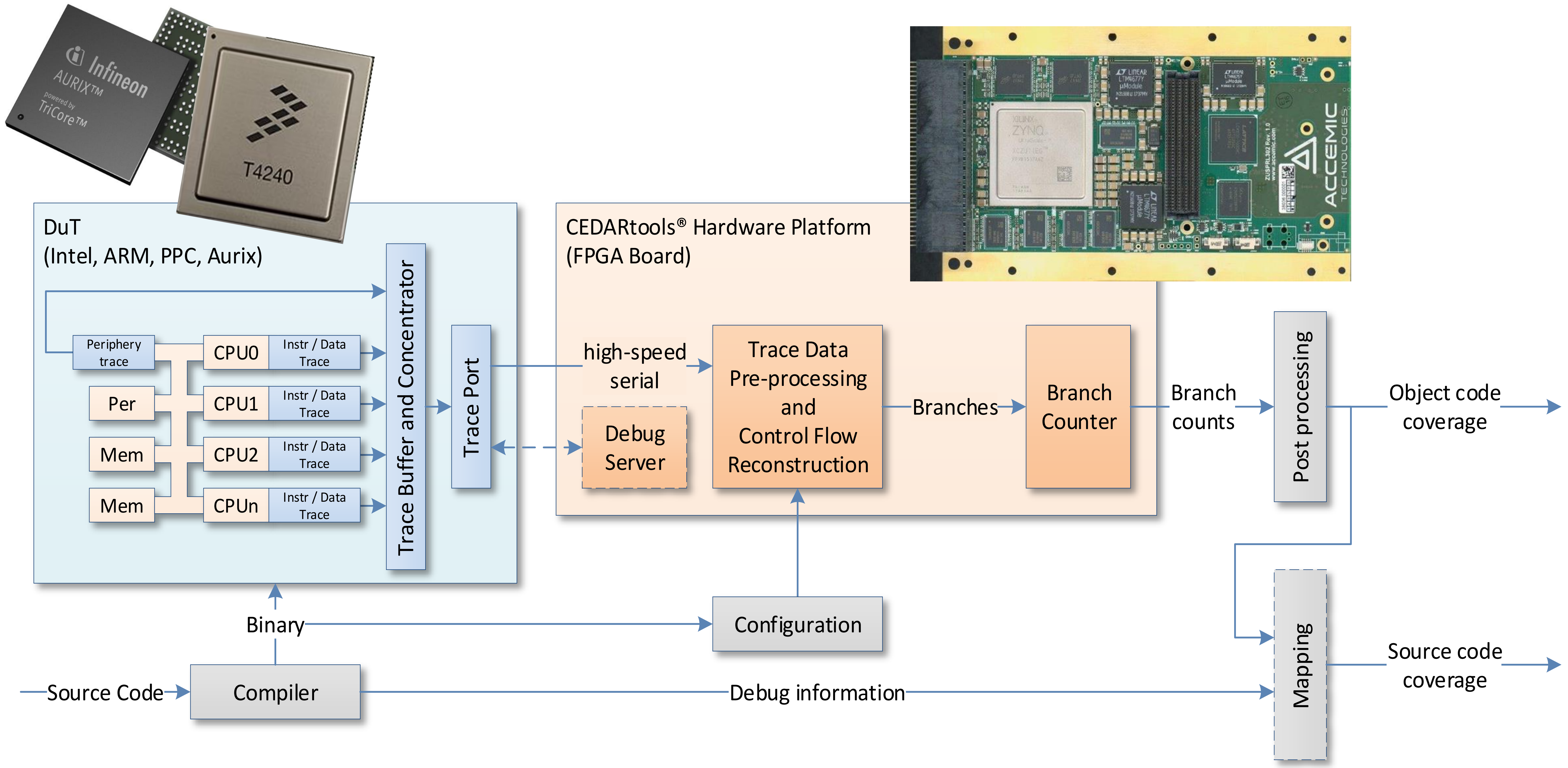
Live Rule Processing





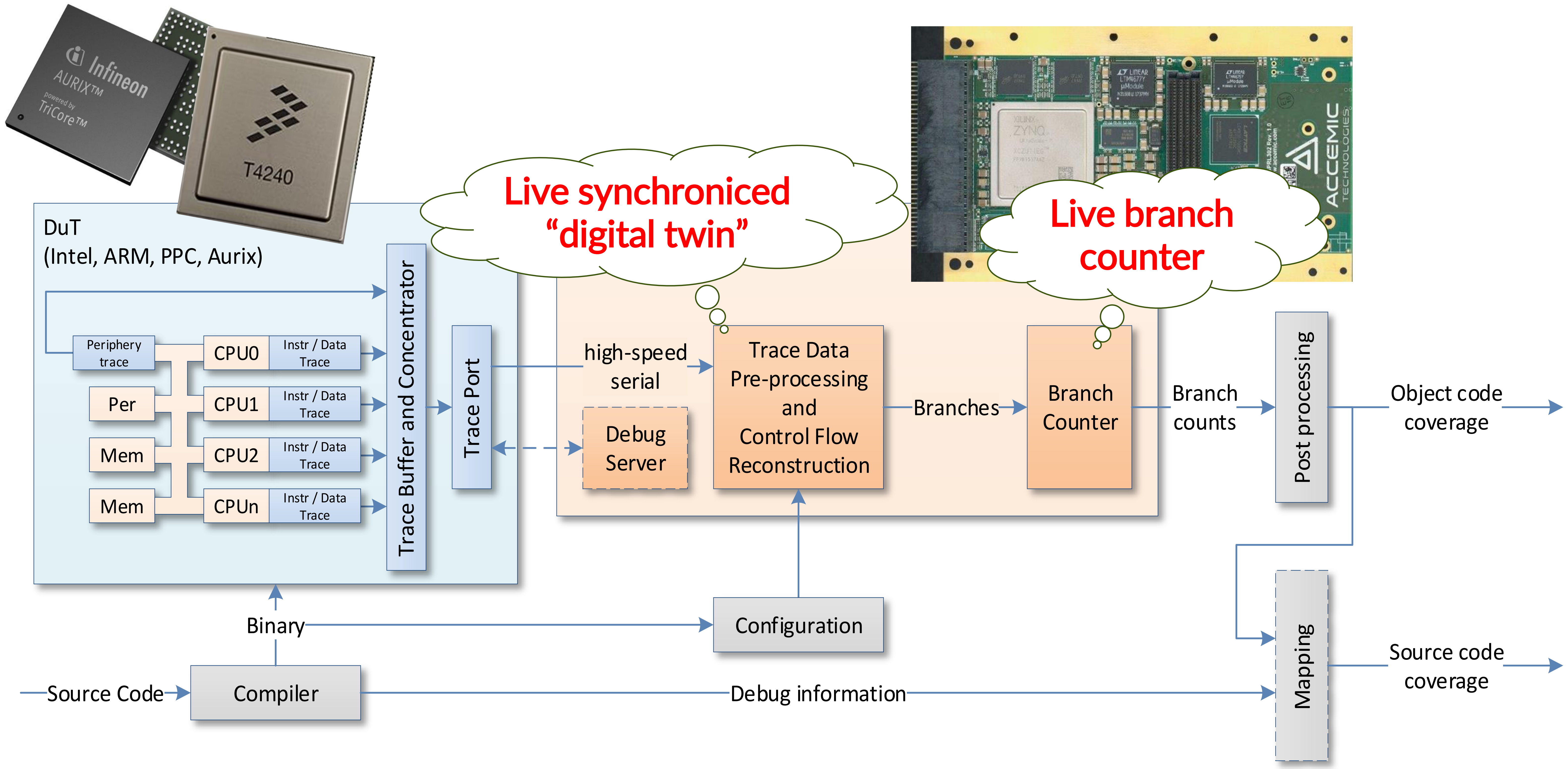
The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Non-intrusive and Continous Observation



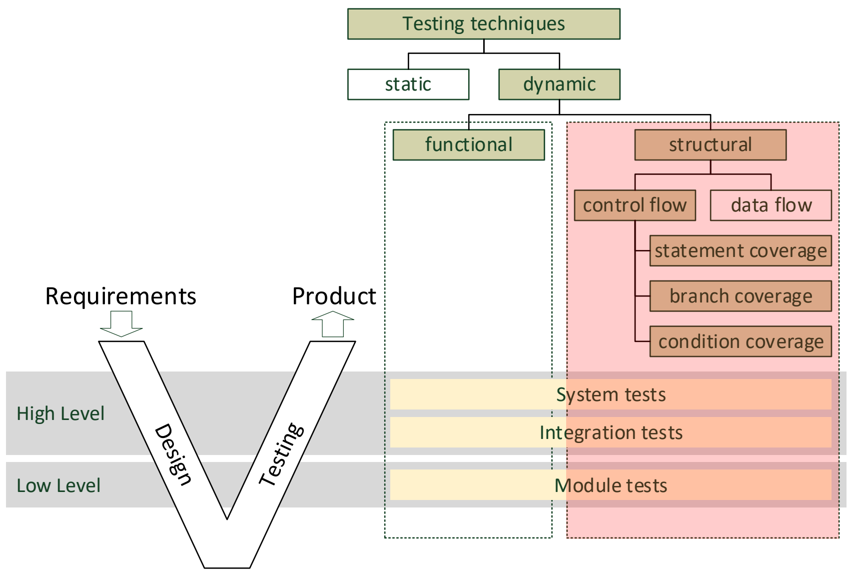
The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Non-intrusive and Continous Observation



The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests



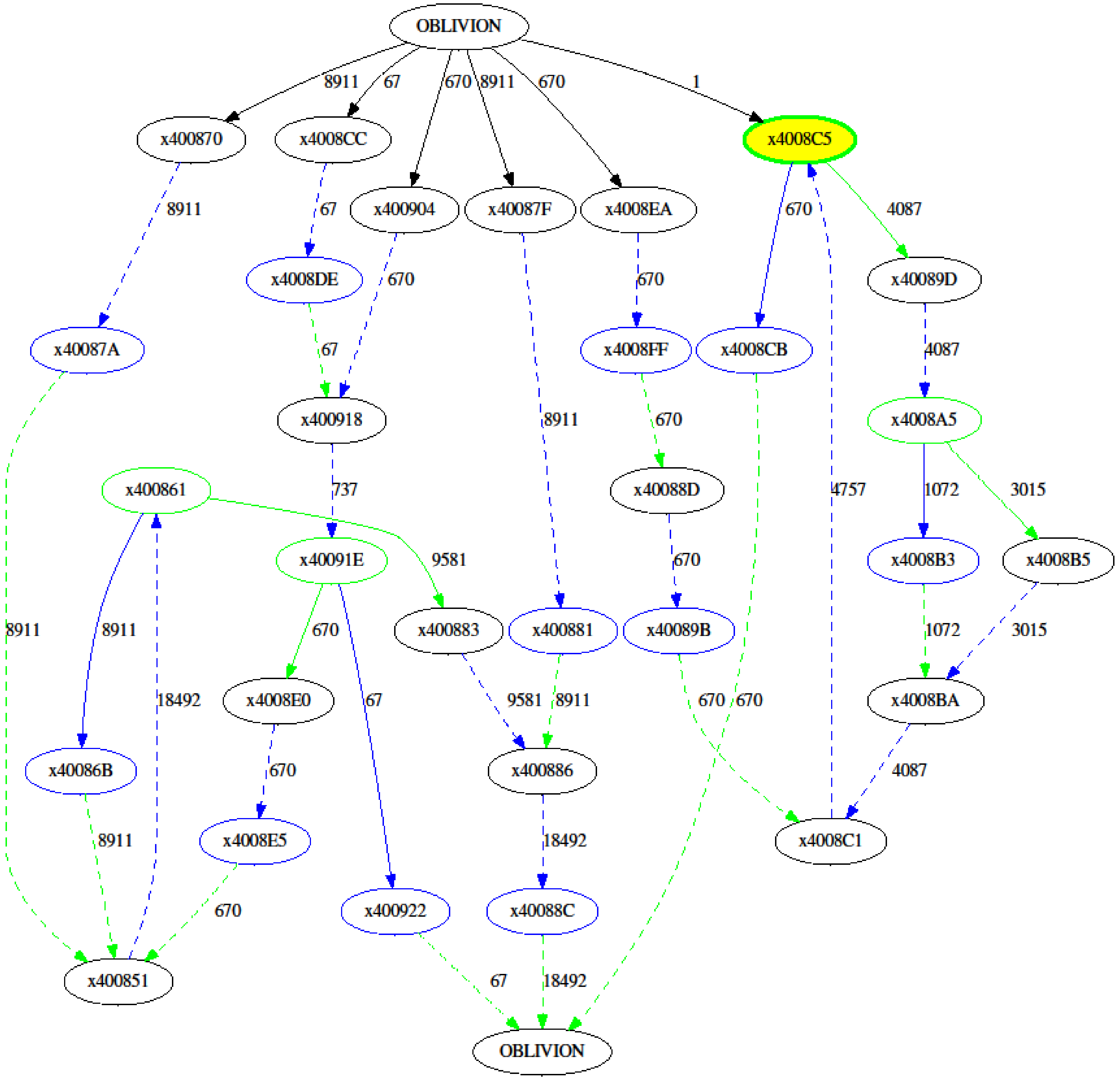
Continuous and non-intrusive

- Statement Coverage
- Branch Coverage (EX/NEX)
- Performance measurement (count executed instructions)

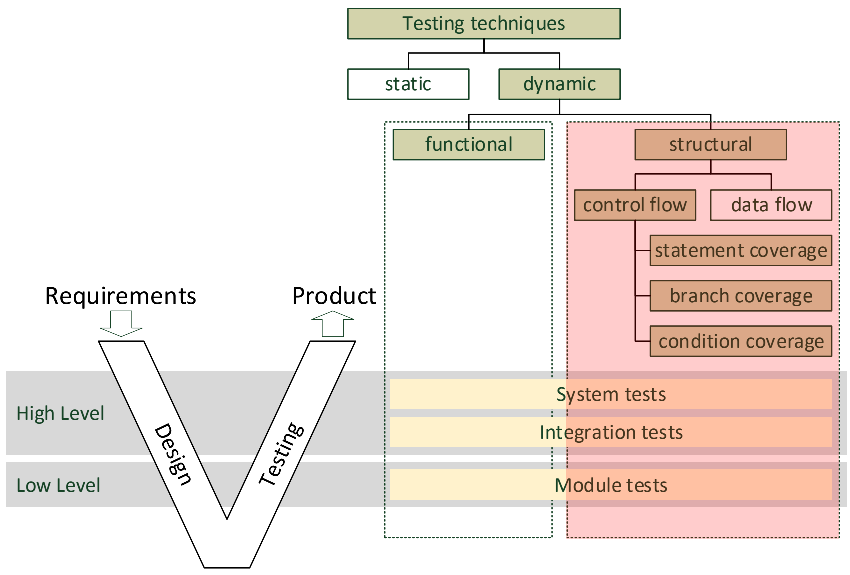
➢ Measured on **object code level**
➢ Measured on **release code**
➢ **No instrumentation**
➢ **No limitation due to trace buffer size**

Targets:

- Infineon Aurix™ (TC2xx & TC3xx),
- Arm® Cortex® -A9,
- Intel® x86 (Atom® E3950),
- NXP QorIQ® P- and T-series,
- Arm® Cortex® -A5x under development.



The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.



High-level Structural Tests

Continuous and non-intrusive

- Statement Coverage
- Branch Coverage (EX/NEX)
- Performance measurement (count executed instructions)

➢ Measured on **object code level**
➢ Measured on **release code**
➢ **No instrumentation**
➢ **No limitation due to trace buffer size**

Targets:

- Infineon Aurix™ (TC2xx & TC3xx),
- Arm® Cortex® -A9,
- Intel® x86 (Atom® E3950),
- NXP QorIQ® P- and T-series,
- Arm® Cortex® -A5x under development.

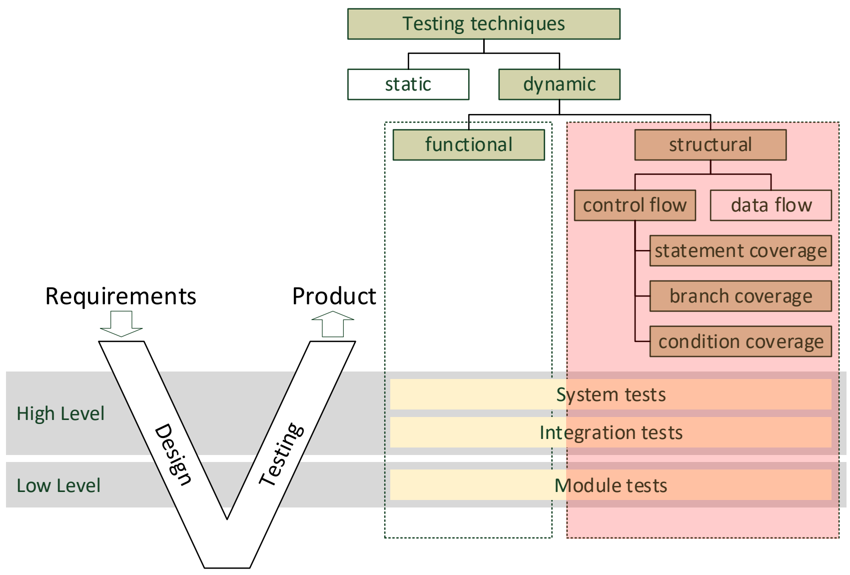
```
        70   10062C: e51b3010 ldr r3, [fp, #-16]
        70   100630: e3530001 cmp r3, #1
[+,+]   70   100634: 9a000010 bls 10067c <_Z13collatz_depthj+0x68>
        61   100678: eaffffeb b 10062c <_Z13collatz_depthj+0x18>

        61   100638: e51b3010 ldr r3, [fp, #-16]
        61   10063C: e2033001 and r3, r3, #1
        61   100640: e3530000 cmp r3, #0
[+,+]   61   100644: 0a000005 beq 100660 <_Z13collatz_depthj+0x4c>
        16   100648: e51b2010 ldr r2, [fp, #-16]
        16   10064C: e1a03002 mov r3, r2
        16   100650: e1a03083 lsl r3, r3, #1
        16   100654: e0833002 add r3, r3, r2
        16   100658: e2833001 add r3, r3, #1
        16   10065C: ea000001 b 100668 <_Z13collatz_depthj+0x54>
        45   100660: e51b3010 ldr r3, [fp, #-16]
        45   100664: e1a030a3 lsr r3, r3, #1
        61   100668: e50b3010 str r3, [fp, #-16]
```

ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests



Continuous and non-intrusive

- Statement Coverage
- Branch Coverage (EX/NEX)
- Performance measurement (count executed instructions)

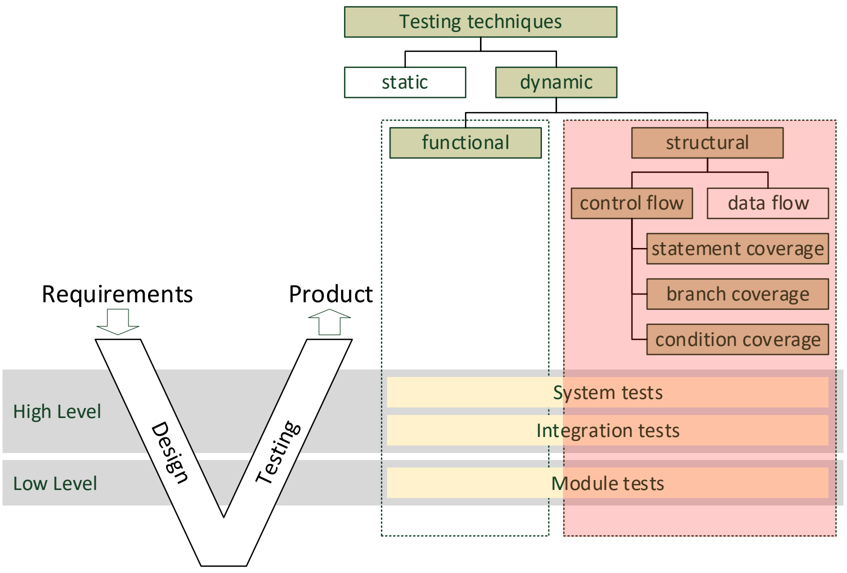
➢ Measured on **object code level**
➢ Measured on **release code**
➢ **No instrumentation**
➢ **No limitation due to trace buffer size**

Targets:

- Infineon Aurix™ (TC2xx & TC3xx),
- Arm® Cortex® -A9,
- Intel® x86 (Atom® E3950),
- NXP QorIQ® P- and T-series,
- Arm® Cortex® -A5x under development

		// Compute n-th Fibonacci number using recursion.
		// - n < 2 does not trigger the else branch.
	275	unsigned fib(unsigned const n) {
[+,+]	275	return (n < 2)? n : fib(n-2) + fib(n-1);
	275	}
		// Unfold Collatz sequence and return its length.
		// - n <= 1 will not execute the while loop at all.
		// - n = 2^k will never trigger the 3*n+1 path.
	9	unsigned collatz_depth(unsigned n) {
	9	unsigned depth = 0;
[+,+]	70	while(n > 1) {
	70	*10062C: e51b3010 ldr r3, [fp, #-16]*
	70	*100630: e3530001 cmp r3, #1*
[+,+]	70	*100634: 9a000010 bls 10067c <_Z13collatz_depthj+0x68>*
	61	*100678: eaffffeb b 10062c <_Z13collatz_depthj+0x18>*
[+,+]	61	n = (n&1)? 3*n+1 : n/2;
	61	*100638: e51b3010 ldr r3, [fp, #-16]*
	61	*10063C: e2033001 and r3, r3, #1*
	61	*100640: e3530000 cmp r3, #0*
[+,+]	61	*100644: 0a000005 beq 100660 <_Z13collatz_depthj+0x4c>*
	16	*100648: e51b2010 ldr r2, [fp, #-16]*
	16	*10064C: e1a03002 mov r3, r2*
	16	*100650: e1a03083 lsl r3, r3, #1*
	16	*100654: e0833002 add r3, r3, r2*
	16	*100658: e2833001 add r3, r3, #1*
	16	*10065C: ea000001 b 100668 <_Z13collatz_depthj+0x54>*
	45	*100660: e51b3010 ldr r3, [fp, #-16]*
	45	*100664: e1a030a3 lsr r3, r3, #1*
	61	*100668: e50b3010 str r3, [fp, #-16]*
	61	depth++;
		}
	9	return depth;
	9	}

High-level Structural Tests



Continuous and non-intrusive

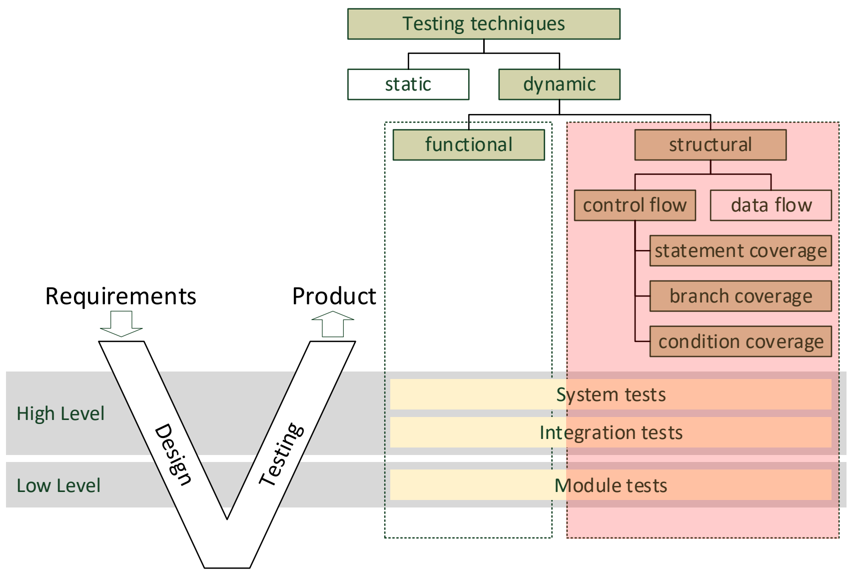
- Statement Coverage
- Branch Coverage (EX/NEX)
- Performance measurement (count executed instructions)

➢ Measured on **object code level**
➢ Measured on **release code**
➢ **No instrumentation**
➢ **No limitation due to trace buffer size**

		`// Compute n-th Fibonacci number using recursion.`
		`// - n < 2 does not trigger the else branch.`
	275	`unsigned fib(unsigned const n) {`
[+,+]	275	`return (n < 2)? n : fib(n-2) + fib(n-1);`
	275	`}`
		`// Unfold Collatz sequence and return its length.`
		`// - n <= 1 will not execute the while loop at all.`
		`// - n = 2^k will never trigger the 3*n+1 path.`
	9	`unsigned collatz_depth(unsigned n) {`
	9	`unsigned depth = 0;`
[+,+]	70	`while(n > 1) {`
[+,+]	61	`n = (n&1)? 3*n+1 : n/2;`
	61	`depth++;`
		`}`
	9	`return depth;`
	9	`}`

Targets:

- Infineon Aurix™ (TC2xx & TC3xx),
- Arm® Cortex® -A9,
- Intel® x86 (Atom® E3950),
- NXP QorIQ® P- and T-series,
- Arm® Cortex® -A5x under development.



High-level Structural Tests

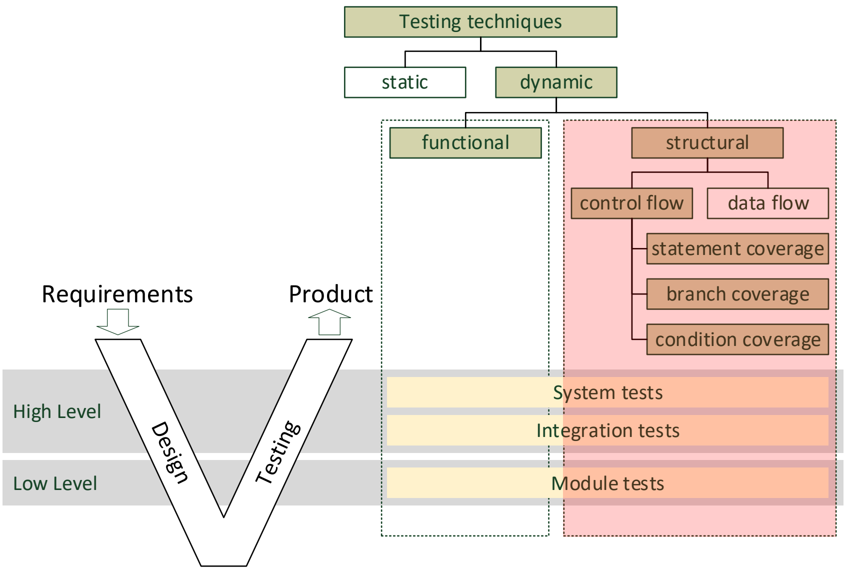
Substitution of Low-Level Structural Tests

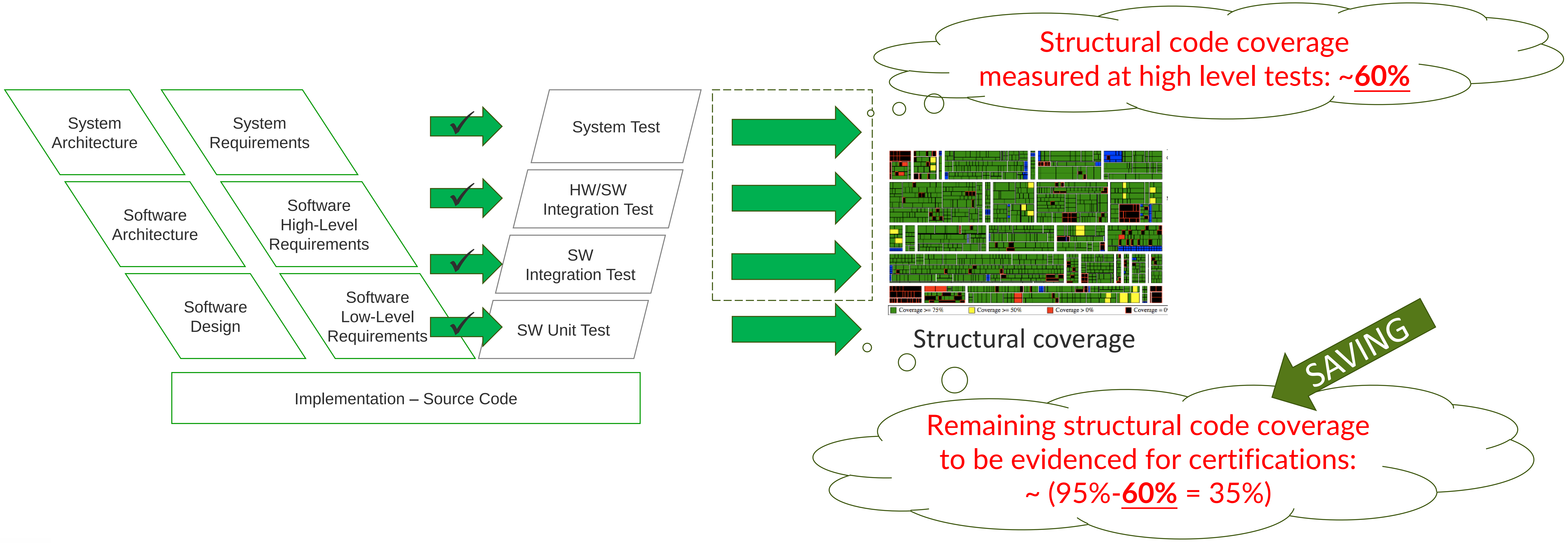
DO178C (6.4):

- If a test case and its corresponding test procedure are developed and executed for HW/SW or SW integration testing and satisfy the requirements-based coverage and structural coverage, it is not necessary to duplicate the test for low-level testing.
- Substituting nominally equivalent low-level tests for high-level tests may be less effective due to the reduced amount of overall functionality tested.

High-level Structural Tests

Substitution of Low-Level Structural Tests

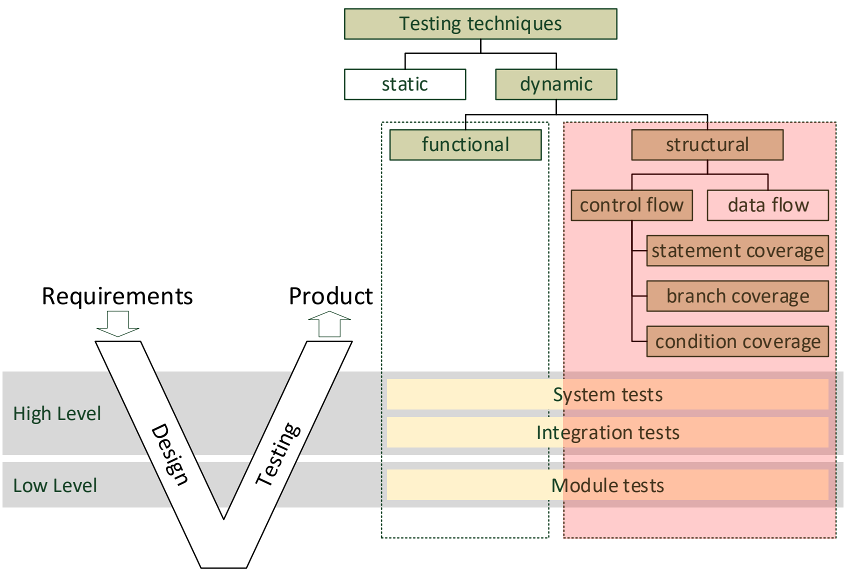


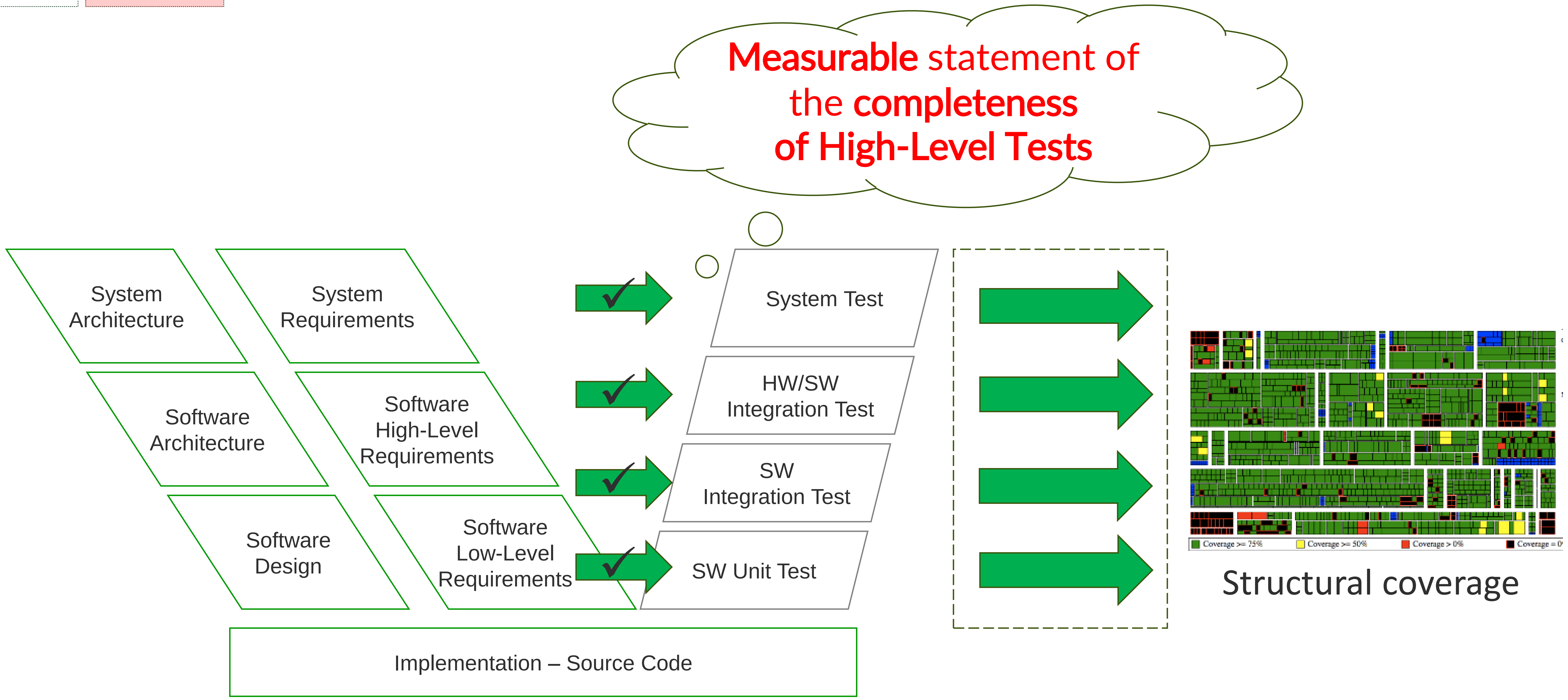


The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests

Completeness of High-Level Tests

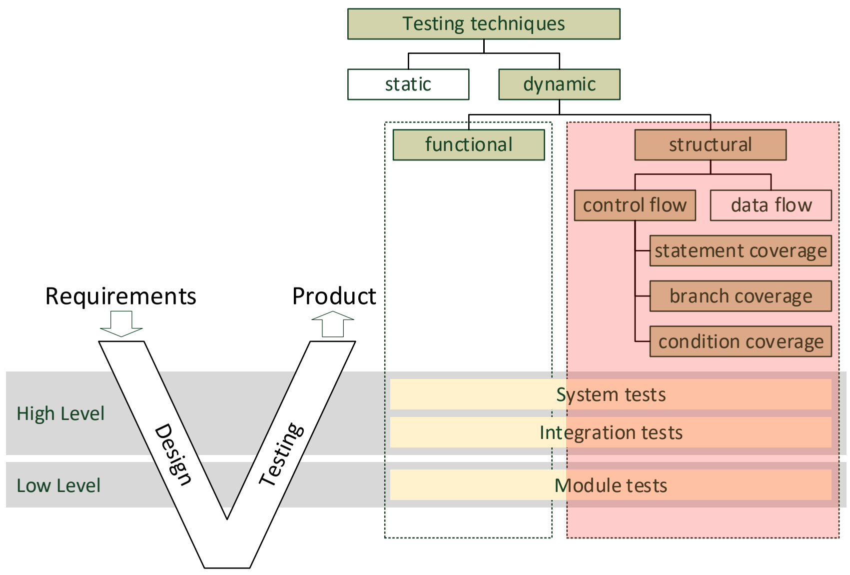




The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests

at Object Code Level



PROs

- It can demonstrate full code coverage at the object code level.
- It can support more “valid” coverage.
- It is closer to the final software .
- It can be implemented with source code programming language independence.
- It can reduce time-consuming manual analysis.
- No instrumentation is required.
- It can also be used for the objective measurement of the quality of integration and system tests.
- It can reduce the test effort by substituting low-level tests.
- Incomplete requirements and tests are found at the system level.

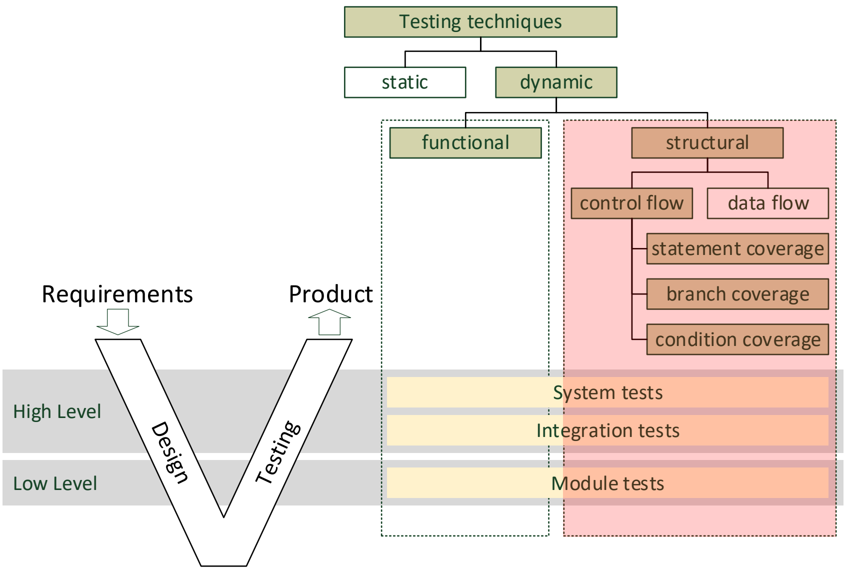
Listing 1: Illustrative example, source code in C

```
char* pass_fail(char grade) {
  static char msg[2][5] = {"pass", "fail"};
  int pass;
  if (grade=='d'||grade=='f') {
      pass = 0;
  } else if (grade=='a' || grade=='b' ||
      grade=='c') {
      pass = 1;
  } else { pass = -1; }
  return pass ? msg[pass] : msg[0];
}
```

Listing 2: *x86* code compiled with -O0

```
/* if (grade == 'd' || grade == 'f') */
8048439: cmpb  $0x64,-0x14(%ebp)
804843d: je    8048445 // jump if grade=='d'
804843f: cmpb  $0x66,-0x14(%ebp)
8048443: jne   804844e // jump if grade!='f'
8048445: movl  $0x0,-0x4(%ebp) // pass:=0
804844c: jmp   8048470 // jump to return
/* else if (grade=='a'||...||grade=='c') */
804844e: cmpb  $0x61,-0x14(%ebp)
8048452: je    8048460 //jump if grade=='a'
8048454: cmpb  $0x62,-0x14(%ebp)
8048458: je    8048460 //jump if grade=='b'
804845a: cmpb  $0x63,-0x14(%ebp)
804845e: jne   8048469 //jump if grade!='c'
8048460: movl  $0x1,-0x4(%ebp) // pass:=1
8048467: jmp   8048470
/* else        (pass:=-1) */
8048469: movl  $0xffffffff,-0x4(%ebp)
...
```

T. Byun, V. Sharma, S. Rayadurgam, S. McCamant, and M. P. Heimdahl, ‘Toward Rigorous Object-Code Coverage Criteria’, in 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2018, vol. 00, pp. 328–338.



High-level Structural Tests

at Object Code Level

CONs

- Source code to object code traceability can be difficult (depending on compiler support).
- Optimizing compiler can use difficult-to-monitor flags to process multi-conditions. (we are working on solutions…)
- Typical tools use the source code level.

Listing 1: Illustrative example, source code in C

```
char* pass_fail(char grade) {
  static char msg[2][5] = {"pass", "fail"};
  int pass;
  if (grade=='d'||grade=='f') {
      pass = 0;
  } else if (grade=='a' || grade=='b' ||
      grade=='c') {
      pass = 1;
  } else { pass = -1; }
  return pass ? msg[pass] : msg[0];
}
```

Listing 3: *x86* code compiled with -Os

```
8048455: push %ebp
8048456: mov  $0x804a01c,%eax// %eax:=msg[0]
804845b: mov  %esp,%ebp
804845d: mov  0x8(%ebp),%edx // %edx:=grade
/* if (grade == 'd' || grade == 'f') */
8048460: mov  %dl,%cl         // %cl:=grade
// ASCII('d')=0x64, ASCII('f')=0x66,
// 'f'^0xfffd='d', 'd'^0xfffd='d'
8048462: and  $0xfffffffd,%ecx
8048465: cmp  $0x64,%cl // 'd', grade
8048468: je   804847e   // %cl=='d'->return
/* else if (grade=='a'||...||grade=='c') */
/* else */
804846a: sub  $0x61,%edx // %edx=grade-'a'
804846d: cmp  $0x3,%dl   // CF=%edx<3?1:0
8048470: sbb  %eax,%eax  // %eax:=CF?-1:0
8048472: and  $0x2,%eax  // %eax:=CF?2:0
8048475: dec  %eax       // %eax:=CF?1:-1
/* return pass ? msg[pass] : msg[0]; */
// %eax:=5*%eax
8048476: imul $0x5,%eax,%eax
// %eax:=msg+%eax
8048479: add  $0x804a01c,%eax
804847e: ...
```

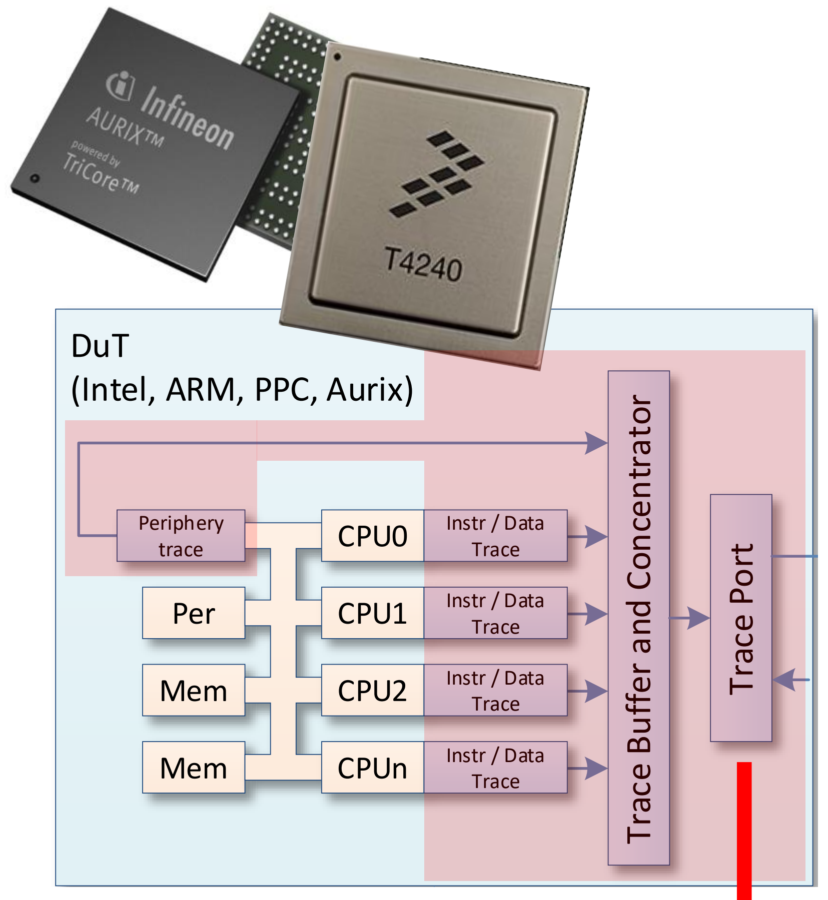
T. Byun, V. Sharma, S. Rayadurgam, S. McCamant, and M. P. Heimdahl, 'Toward Rigorous Object-Code Coverage Criteria', in 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2018, vol. 00, pp. 328–338.

ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

High-level Structural Tests

Key enabler: Processor Trace



Most processors provide trace infrastructure:

- ARM Cortex-A/-M/-R: CoreSight architecture
- Intel x86: IntelPT
- NXP QorIQ P-series, T-series: Debug Assist Module
- Infineon Aurix: Emulation Device

ARM CoreSight Architecture Specification

Intel® 64 and IA-32 Architectures Software Developer's Manual

T4240R2 Advanced QorIQ Debug and Performance Monitoring Reference Manual

TC29/7/6/3xED

32-Bit Single-Chip Micocontroller

Trace date consists of high-bandwidth information for reconstruction of

- Control flow,
- OS related events (task changes),
- Data access (address, value),

  and hardware-supported instrumentation: printf() replacement by simple MOV command.

Hardware-based monitoring infrastructure is integrated in most processors – *and is already paid by you …*

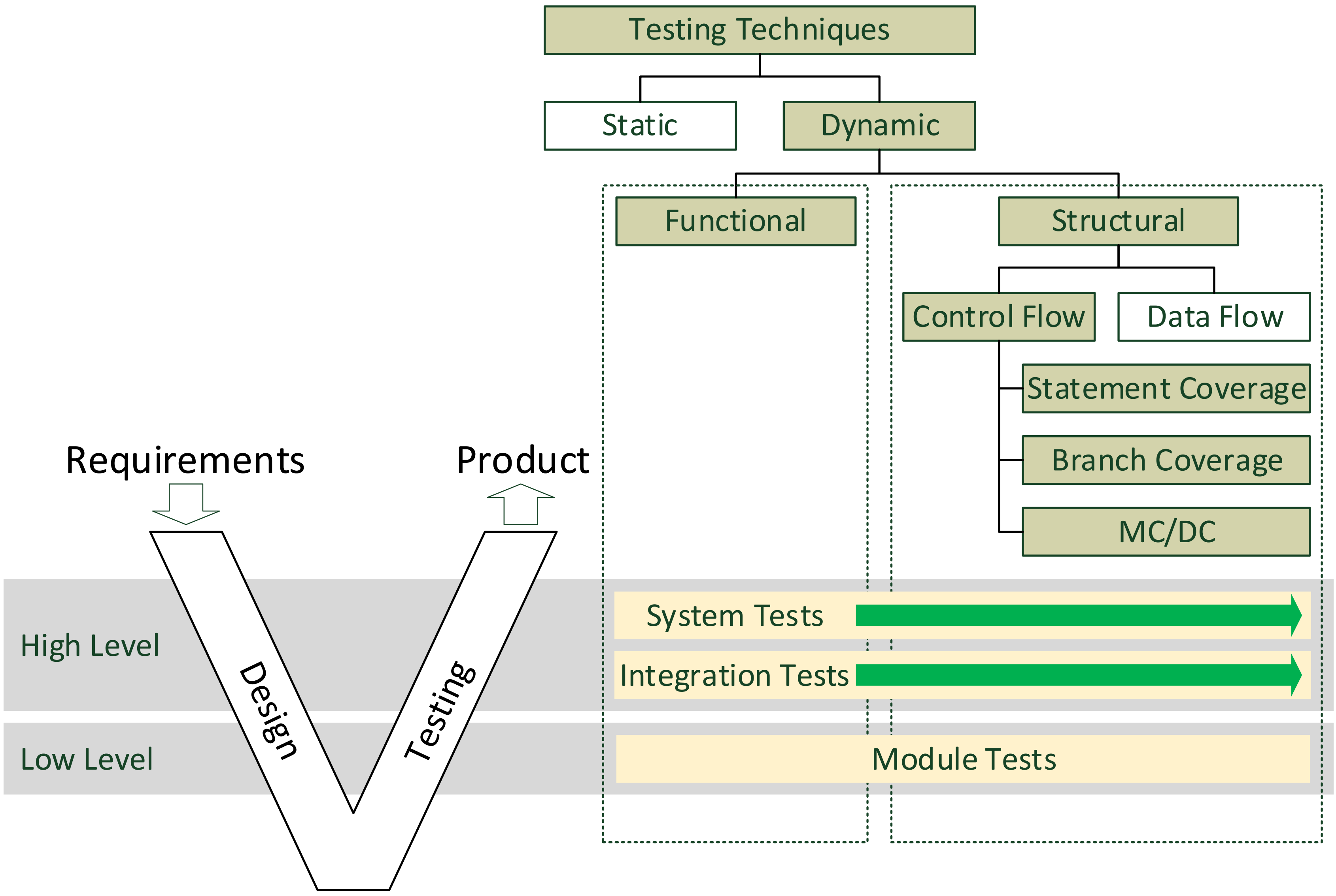
Mind trace interface access opportunities:

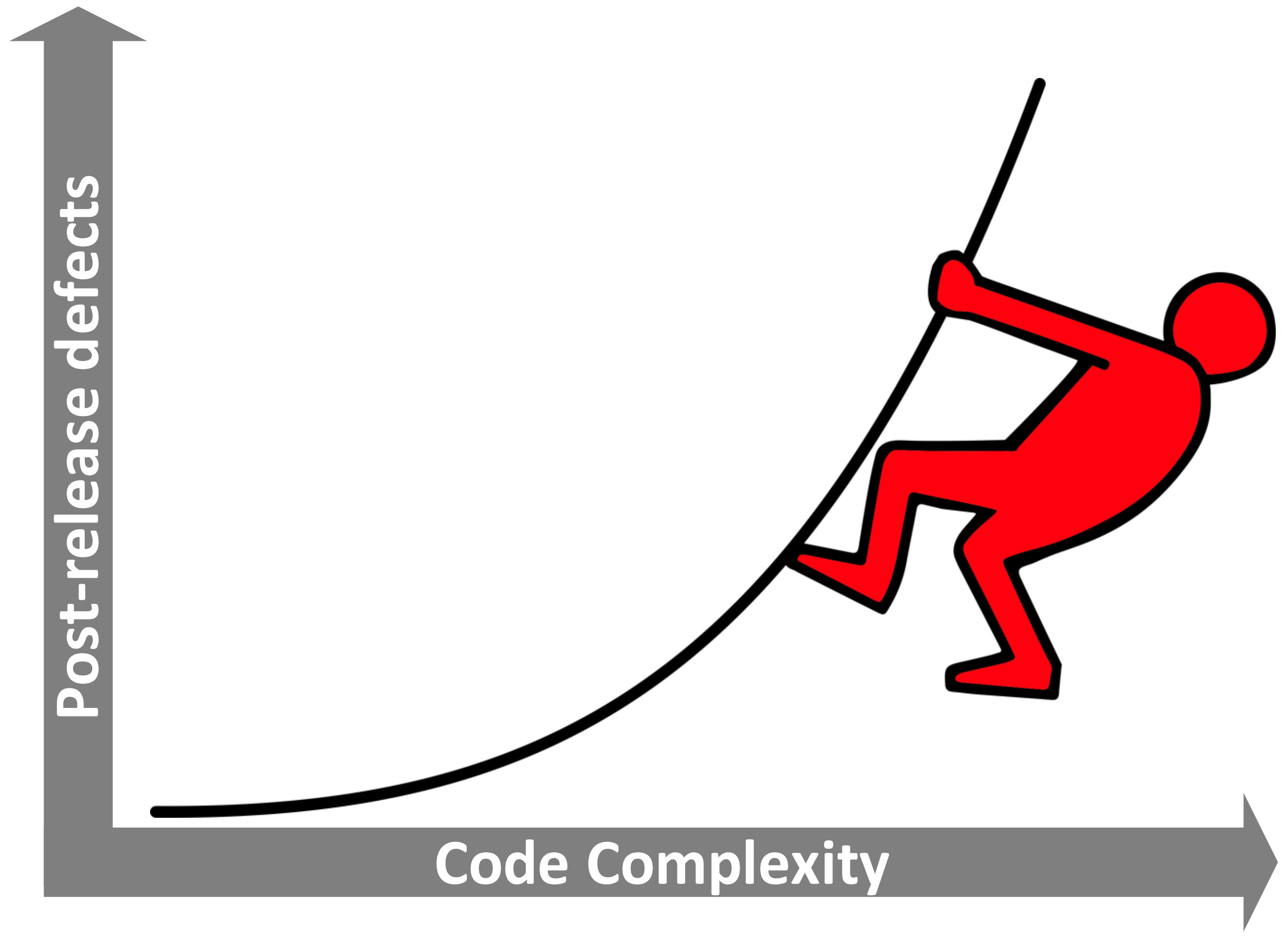
- in your hardware system requirement specifications and

- in your SoC architecture decision!

Bug-free Software is a Myth.

But there's something we can do.





Contact:
Accemic Technologies GmbH
Franz-Huber-Str. 39
83088 Kiefersfelden
www.accemic.com

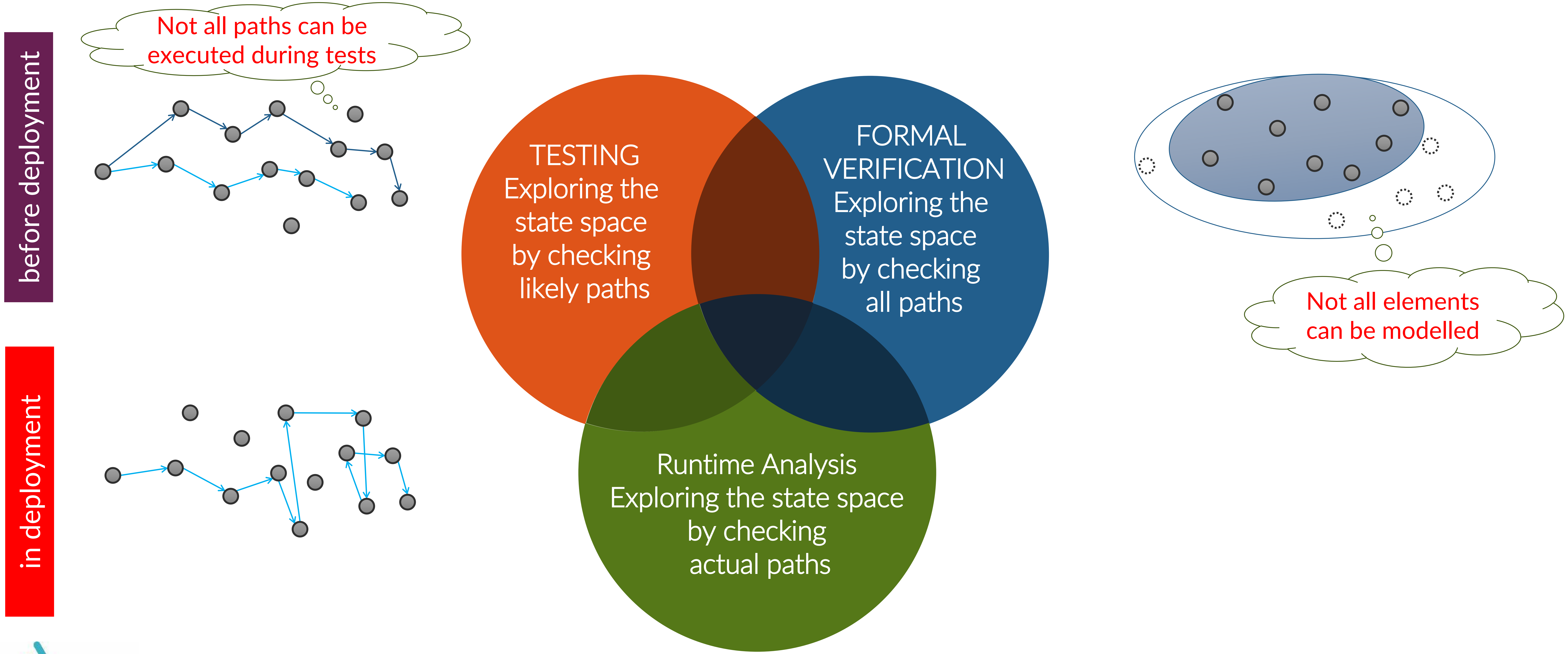
Dr.-Ing. Alexander Weiss
aweiss@accemic.com
+49 8033 6039795

ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Backup slides

Runtime Analysis

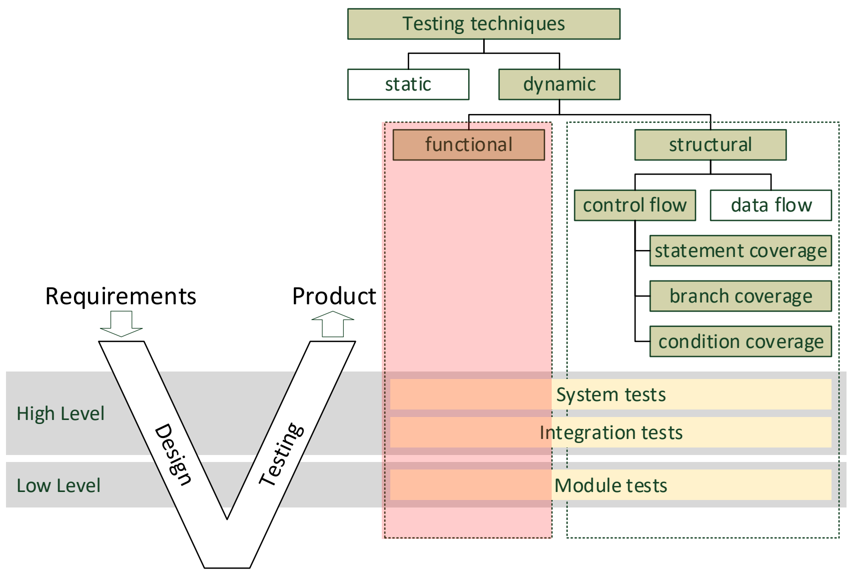


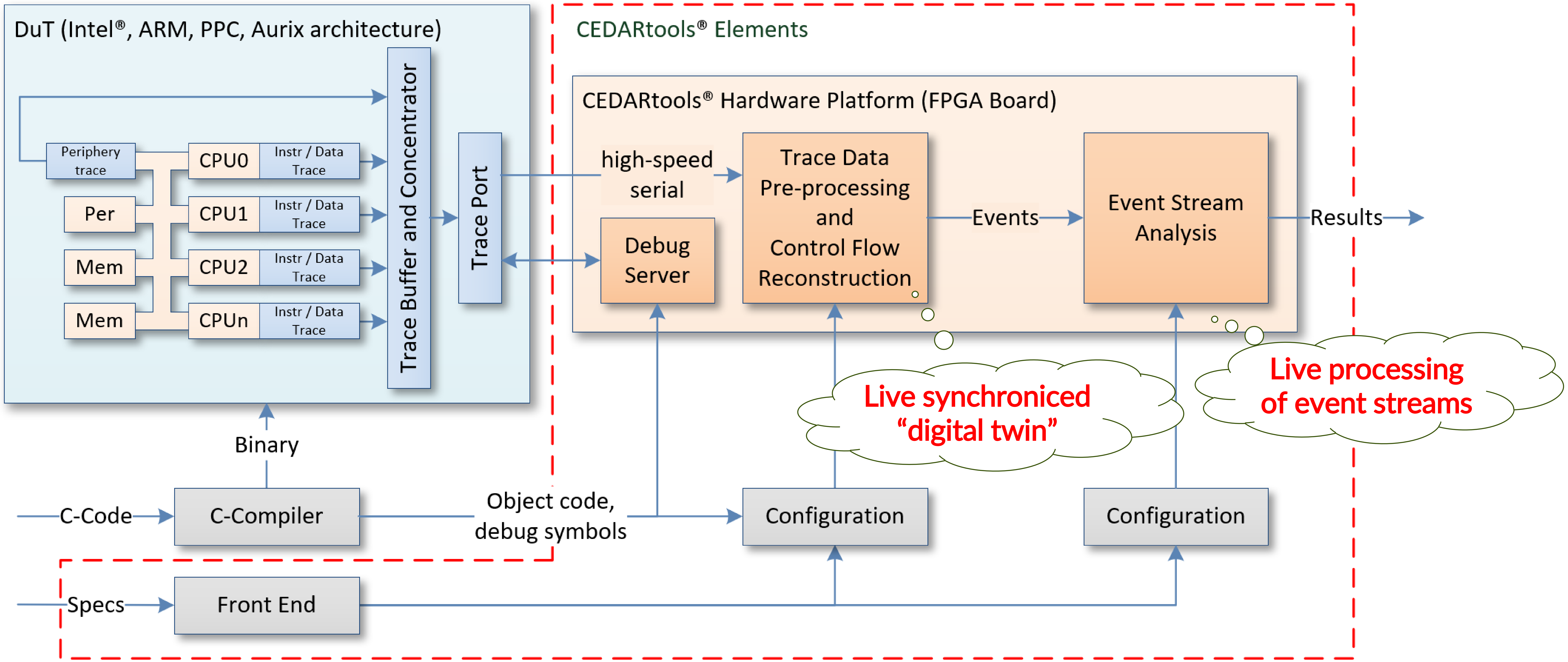
ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Runtime Analysis

Timing Analysis

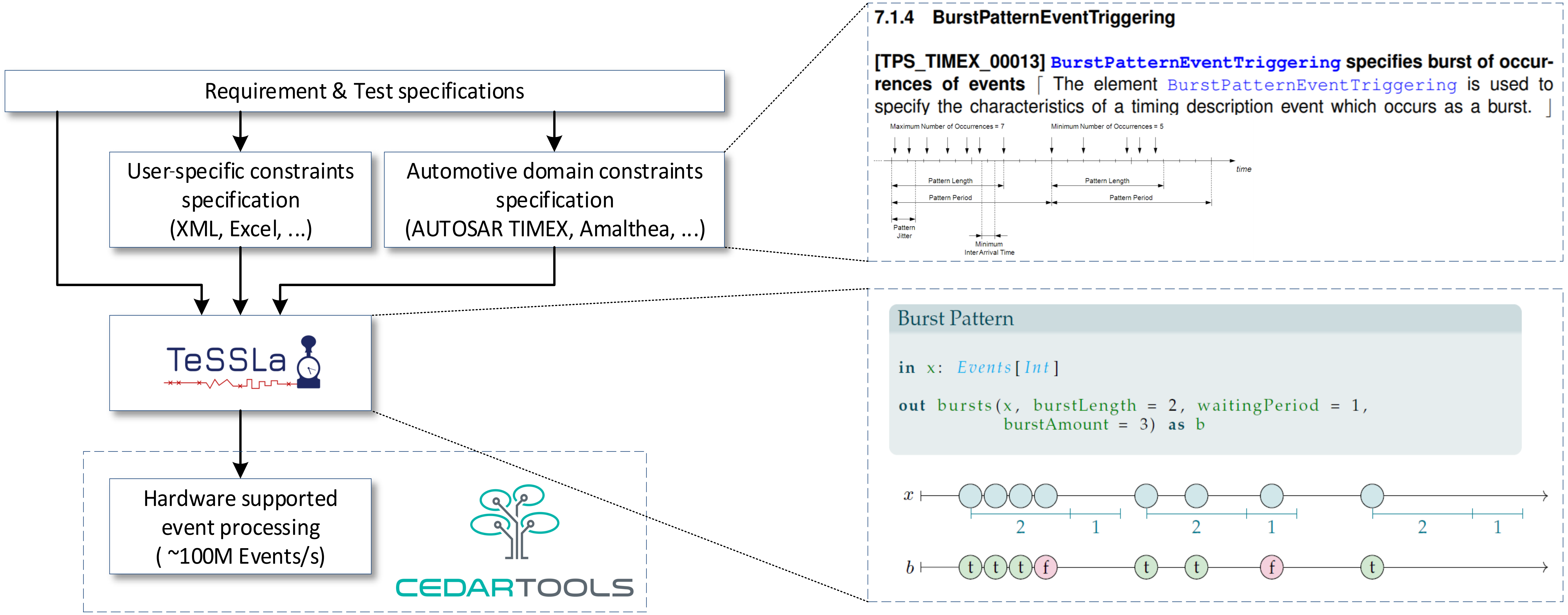




The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Runtime Analysis

Timing Analysis



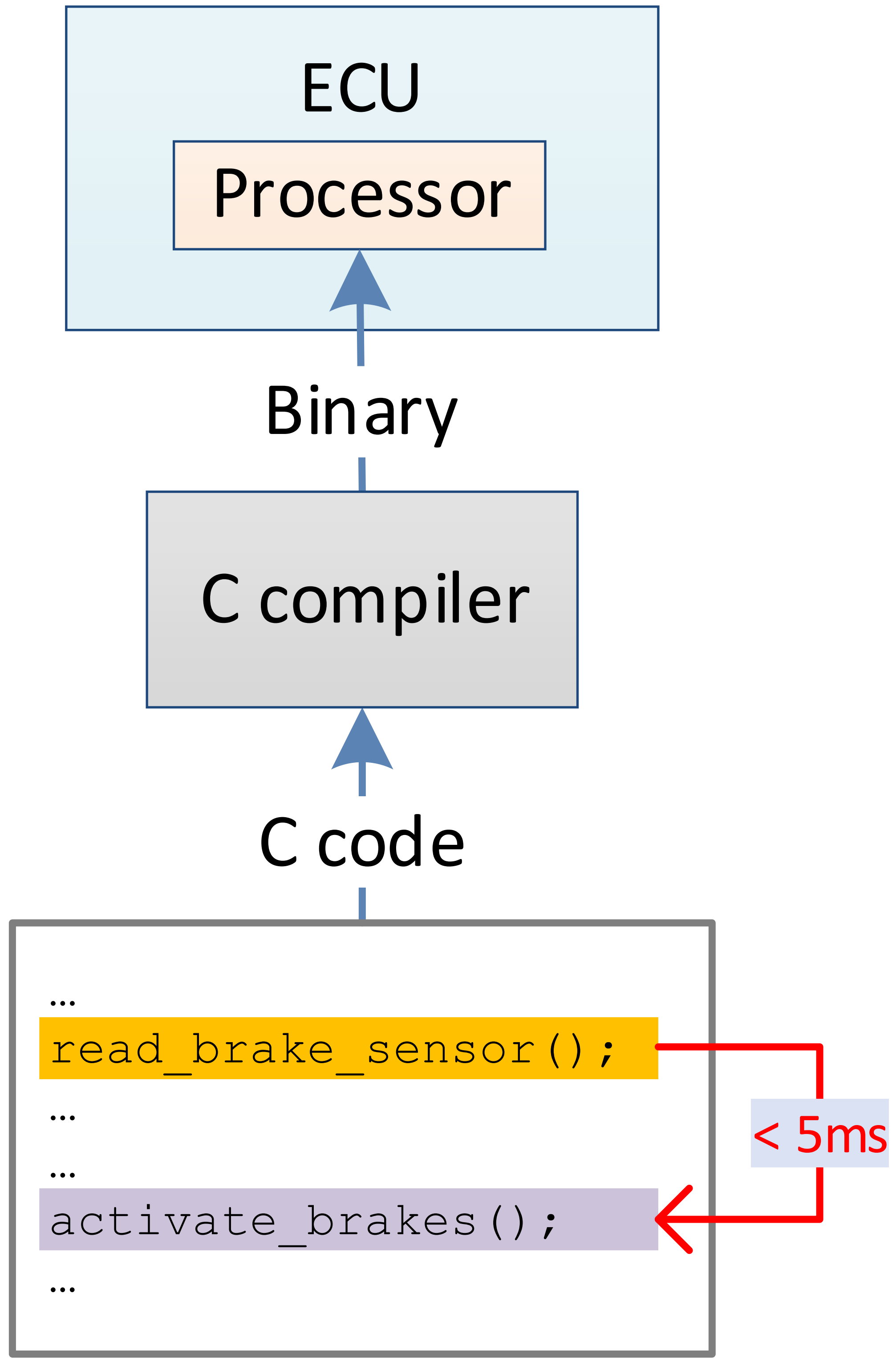
- High-level specification language
- Hardware-supported event processing
- Multiple constraints can be checked in parallel

(see tessla.io)

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Runtime Analysis

Timing Analysis

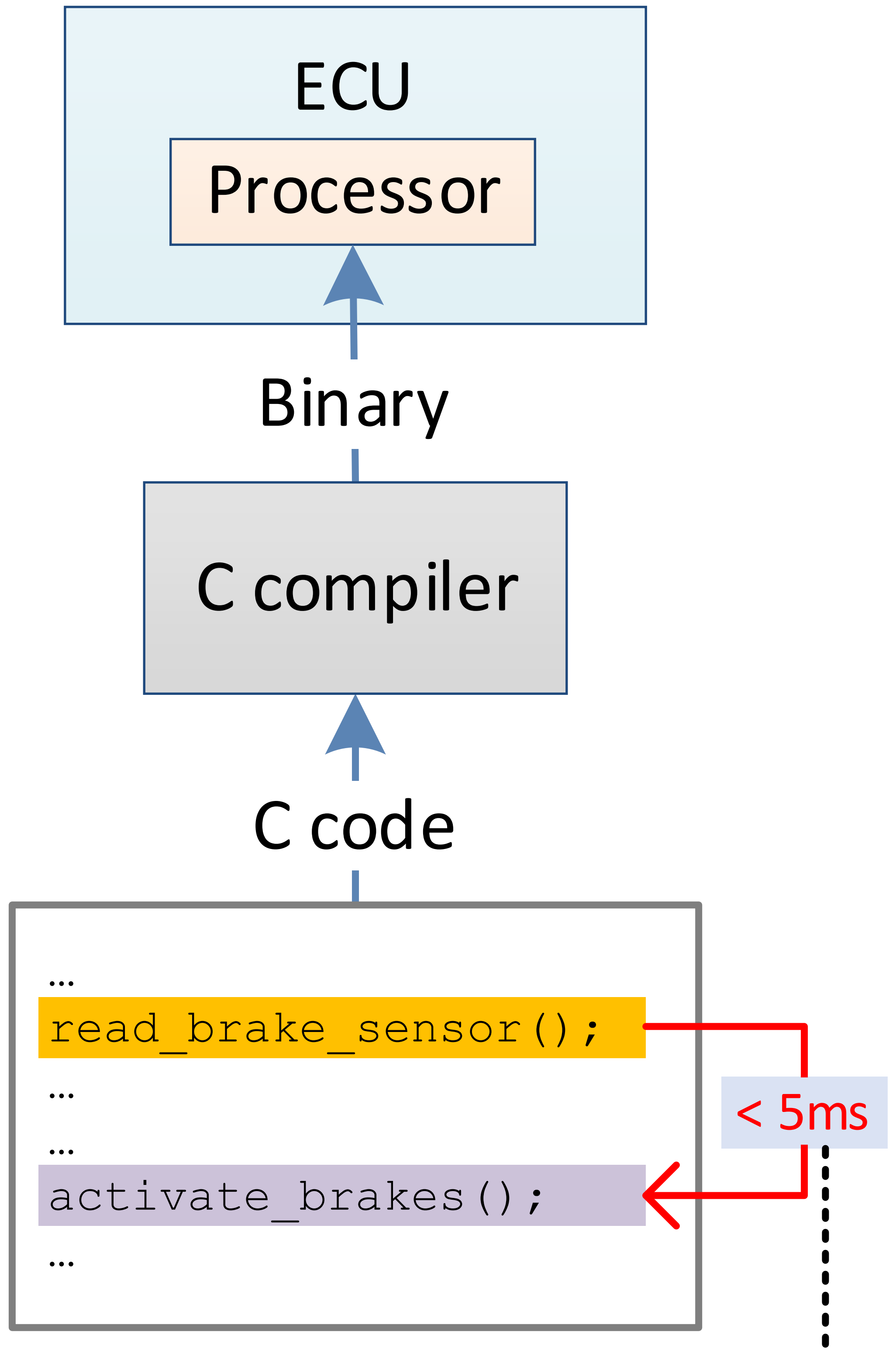


- Use case: Safety-critical application to control brakes
- Requirement:
  - Ensure Timing Constraint from **pressing the brakes**, until their **activation**
  - Constraint: Should react within 5ms!

Runtime Analysis

Timing Analysis

- Use case: Safety-critical application to control brakes
- Requirement:
  - Ensure Timing Constraint from **pressing the brakes**, until their **activation**
  - Constraint: Should react within 5ms!



test specificationen

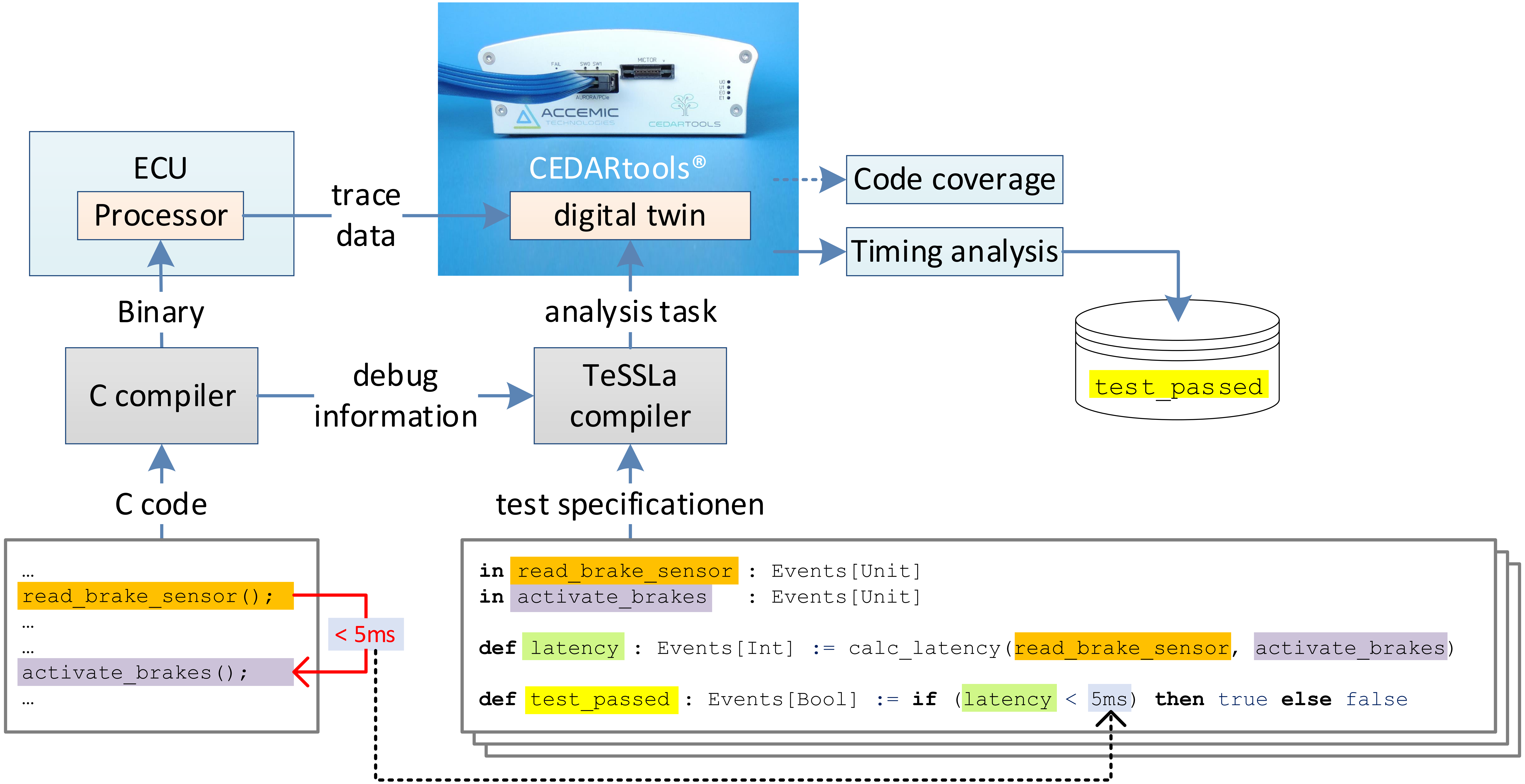
```
in read_brake_sensor : Events[Unit]
in activate_brakes   : Events[Unit]

def latency : Events[Int] := calc_latency(read_brake_sensor, activate_brakes)

def test_passed : Events[Bool] := if (latency < 5ms) then true else false
```

Runtime Analysis

Timing Analysis



The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.

Runtime Analysis

Timing Analysis

https://www.youtube.com/watch?v=3AYVWK-X9nw&feature=youtu.be

File Edit Selection View Go Run Terminal Help

demo.cpp | latency-activate-brakes.tessla

demo > demo.cpp > main(int, char * [])

```cpp
/* Program Entry */
int main(int argc,char *argv[])
{
    /* Trace Setup */
    atexit(cedar_kill_trace);
    signal(SIGINT, sigint_handler);
    cedar_linux_init_epu(false);
    cedar_set_appid(0);

    /* Program Routine */
    while(1) {
        run_task();
        usleep(10e3);
    }

    return 0;
}

void run_task()
{
    /* Sample Brake Sensor */
    float brake_angle = read_brake_sensor();

    /* Process */
    int strength;
    strength = calculate_brake_strengh_for_angle(brake_angle);
```

multi-specs* ⊗ 0 ⚠ 0 Zen Ln 38, Col 2 Tab Size: 4 UTF-8 LF C++ Linux Insert

ACCEMIC TECHNOLOGIES

The pioneer in embedded systems dynamic analysis.
Automated, non-intrusive and continuous.