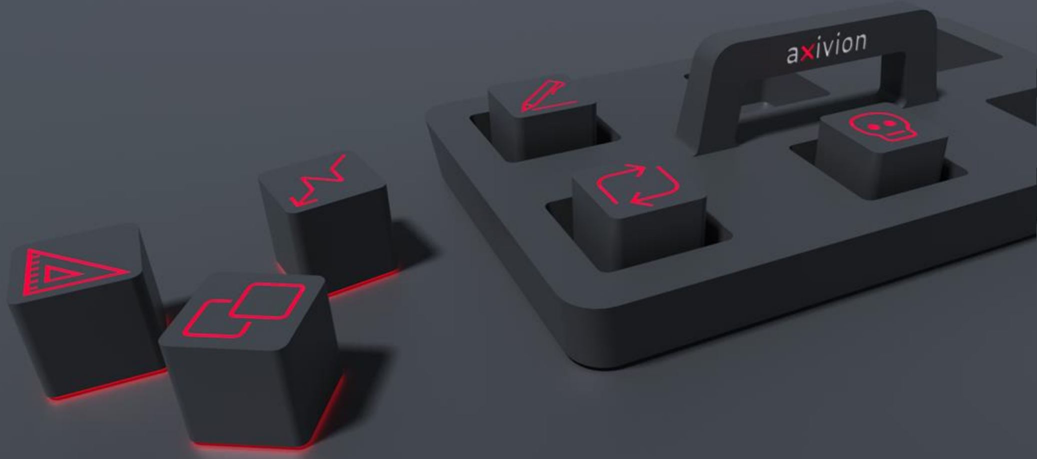
Testing, Model Checking and Static Analysis

Dream Team or Rivals?



Sebastian Krings

Axivion GmbH

axivion

stopping software erosion

> “Software is eating the world.”
>
> \- Marc Andreessen

Ever-increasing Dependence on Safety-Critical Software

axivion
stopping software erosion

- Software used everywhere, from cars to trains to medical devices
- Software errors lead to damages, loss of equipment and have killed people
- Enormous financial losses due to software not even deemed safety-critical

We need to get software right.

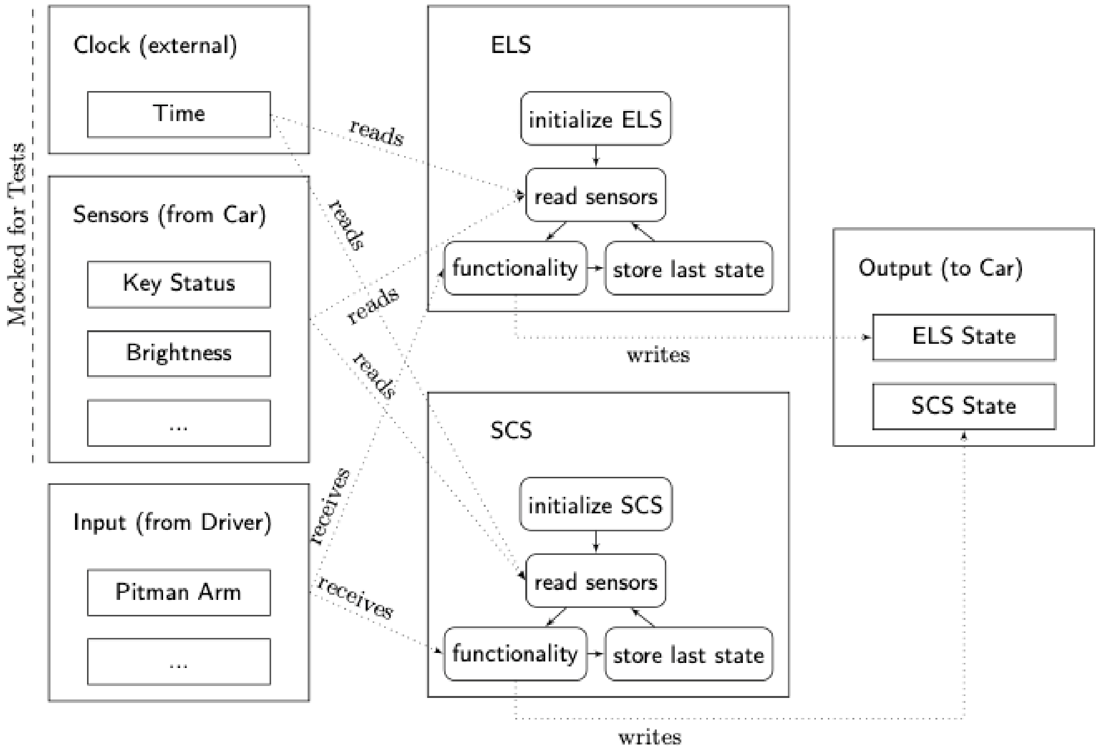
Let’s see how.

Images by Catalina Márquez under CC BY-SA 3.0 and ESA

Case Study - Interacting Assistant Systems

System	Controls	Reads	Development
Adaptive Exterior Light System (ELS)	Head- and taillights • Brightness • Illumination distance • Curve lighting	• Driver input via actuators • Sensors for Speed, etc.	• To be developed in C • Following the MISRA C 2012 coding guidelines • Obviously highly safety critical
Speed Control System (SCS)	Speed of vehicle • Acceleration • Deceleration • Emergency breaks		

Component Diagram

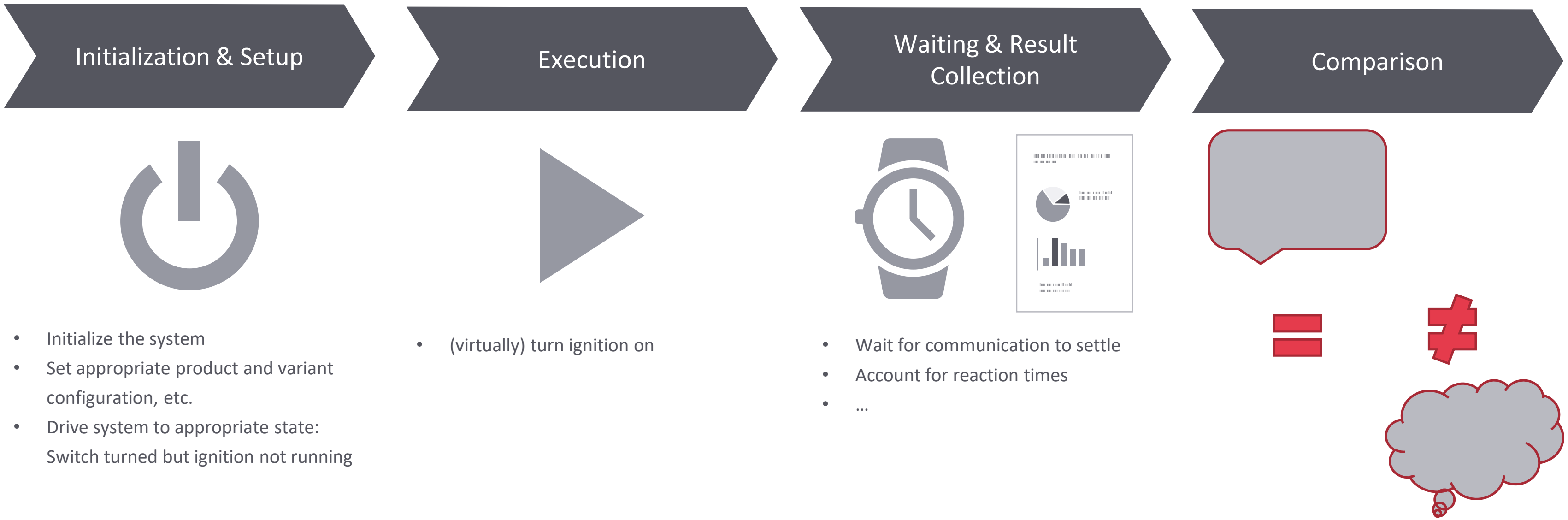


Testing - Overview

- Broadly speaking: make the system under test execute a certain scenario and check if we like the outcome
- Many different kinds and ideas
    - White-, black- and grey-box testing
    - Unit, integration and system testing
    - Sanity- and smoke-testing
    - Manual and automated testing
    - Hardware-in-the-loop and mocking
    - ….

Testing - Example

Requirement: If the ignition is ‘On’ and the light rotary switch is in the position ‘On’, then low beam headlights are activated.



Initialization & Setup

- Initialize the system
- Set appropriate product and variant configuration, etc.
- Drive system to appropriate state: Switch turned but ignition not running

Execution

- (virtually) turn ignition on

Waiting & Result Collection

- Wait for communication to settle
- Account for reaction times
- …

Comparison

Testing - Conclusions and Remarks

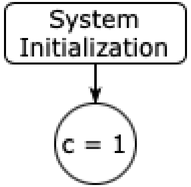
- Requirement is only partially tested
  - We have switch first, ignition second
  - But we are missing ignition first, switch second
- Needed to mock away the sensors
- No hope to ever test exhaustively
- Add some coverage metric
  - Statement, condition, MC/DC, …

Model Checking - Overview

- Broadly speaking: explore all states the system might be in, check an invariant property for each
- Many different kinds and ideas:
  - Abstraction level
  - Explicit or symbolic states
- Consider expressiveness of underlying logic, i.e., what is the invariant allowed to talk about
  - State only
  - Path on which a state was reached
  - Past or future states

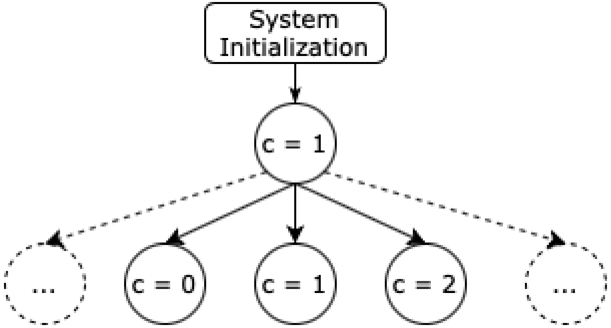
Model Checking - State Spaces

```
extern int some_value();
int main {
       int c = 1;
       c += some_value();
       ...
}
```



Model Checking - State Spaces

```
extern int some_value();
int main {
       int c = 1;
       c += some_value();
       ...
}
```



Model Checking - Example

- Requirement: Whenever the low or high beam headlights are activated, the tail lights are activated, too.
- Translate into invariant
- Check using CBMC (for instance)

```
assert(implies(state.lowBeamLeft > 0,
               state.tailLampLeft > 0 &&
               state.tailLampRight > 0));
```

Model Checking - Example - Resulting Output

```
State 59 file light/light-impl.c line 242 function light_do_step thread 0
----------------------------------------------------
 ks=/*enum*/NoKeyInserted (00000000000000000000000000000000)

State 63 file light/light-impl.c line 242 function light_do_step thread 0
----------------------------------------------------
 ks=/*enum*/KeyInIgnitionOnPosition (00000000000000000000000000000010)

State 65 file light/light-impl.c line 244 function light_do_step thread 0
----------------------------------------------------
 engine_on=FALSE (00000000)

State 69 file light/light-impl.c line 244 function light_do_step thread 0
----------------------------------------------------
 engine_on=TRUE (00000001)
```

Model Checking - Results

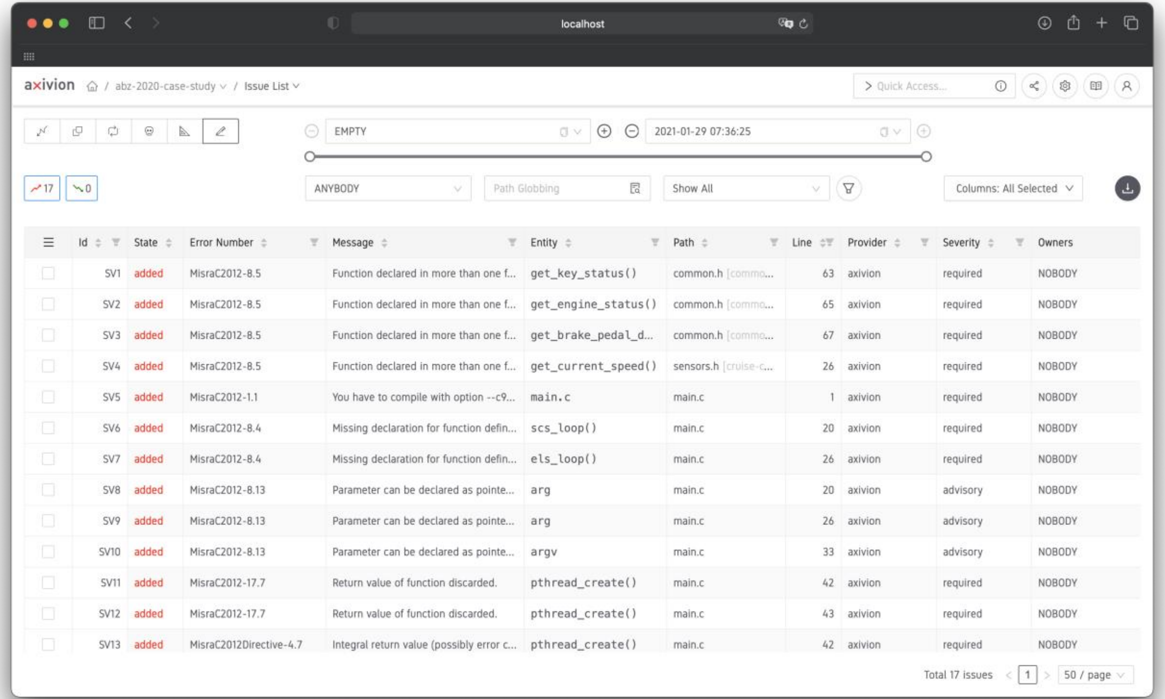
- Combinatorial explosion!
- Checking industrial-scale systems often infeasible
- But checking the whole system is not needed to discover errors
- Techniques to increase applicability:
  - Symbolic techniques
  - Symmetry detection and avoidance
  - …

Static Analysis - Overview

- Broadly speaking: avoid code execution, try to infer properties from the source code
- Many different techniques under one umbrella:
    - Style checking or linting
    - Data flow and pointer-based analyses
    - Abstract interpretation
    - Symbolic techniques
    - ….
- Comes at a price: over- and under-approximations and resulting false positives / negatives

Static Analysis - Axivion - Issue List Example

stopping software erosion



axivion / abz-2020-case-study / Issue List

EMPTY

2021-01-29 07:36:25

ANYBODY

Path Globbing

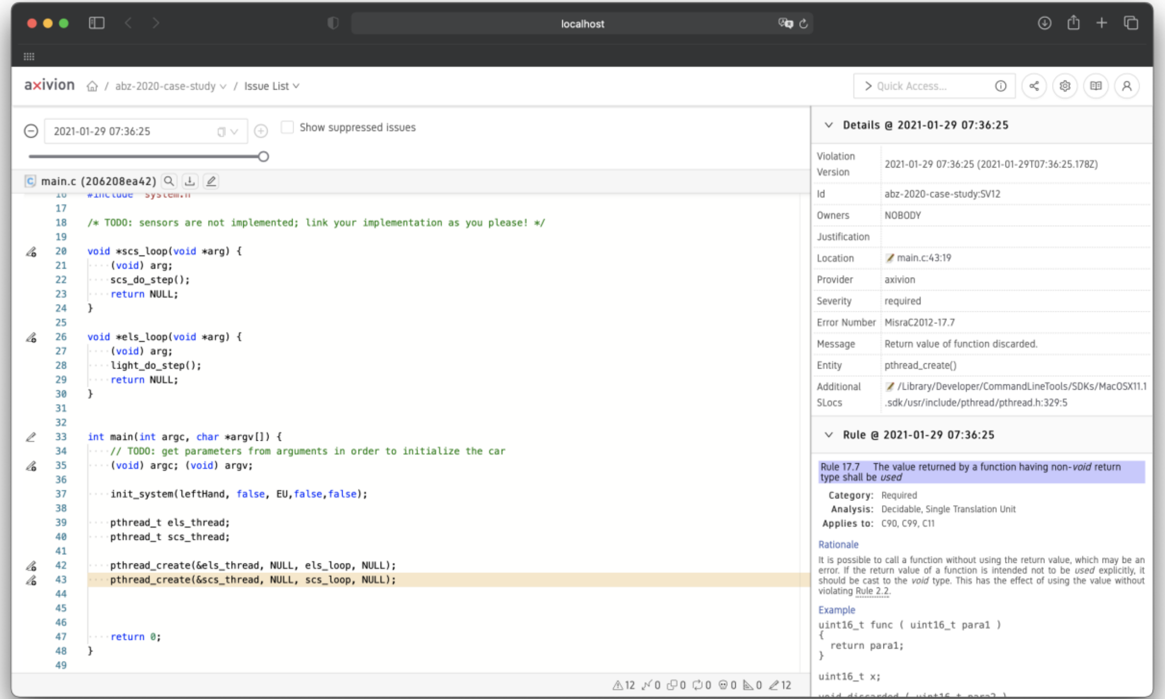
Show All

	Id	State	Error Number	Message	Entity	Path	Line	Provider	Severity	Owners
☐	SV1	added	MisraC2012-8.5	Function declared in more than one f...	`get_key_status()`	common.h [commo...	63	axivion	required	NOBODY
☐	SV2	added	MisraC2012-8.5	Function declared in more than one f...	`get_engine_status()`	common.h [commo...	65	axivion	required	NOBODY
☐	SV3	added	MisraC2012-8.5	Function declared in more than one f...	`get_brake_pedal_d...`	common.h [commo...	67	axivion	required	NOBODY
☐	SV4	added	MisraC2012-8.5	Function declared in more than one f...	`get_current_speed()`	sensors.h [cruise-c...	26	axivion	required	NOBODY
☐	SV5	added	MisraC2012-1.1	You have to compile with option --c9...	`main.c`	main.c	1	axivion	required	NOBODY
☐	SV6	added	MisraC2012-8.4	Missing declaration for function defin...	`scs_loop()`	main.c	20	axivion	required	NOBODY
☐	SV7	added	MisraC2012-8.4	Missing declaration for function defin...	`els_loop()`	main.c	26	axivion	required	NOBODY
☐	SV8	added	MisraC2012-8.13	Parameter can be declared as pointe...	`arg`	main.c	20	axivion	advisory	NOBODY
☐	SV9	added	MisraC2012-8.13	Parameter can be declared as pointe...	`arg`	main.c	26	axivion	advisory	NOBODY
☐	SV10	added	MisraC2012-8.13	Parameter can be declared as pointe...	`argv`	main.c	33	axivion	advisory	NOBODY
☐	SV11	added	MisraC2012-17.7	Return value of function discarded.	`pthread_create()`	main.c	42	axivion	required	NOBODY
☐	SV12	added	MisraC2012-17.7	Return value of function discarded.	`pthread_create()`	main.c	43	axivion	required	NOBODY
☐	SV13	added	MisraC2012Directive-4.7	Integral return value (possibly error c...	`pthread_create()`	main.c	42	axivion	required	NOBODY

Total 17 issues < 1 > 50 / page

Static Analysis - Axivion - Issue Details Example

axivion
stopping software erosion



```
17
18  /* TODO: sensors are not implemented; link your implementation as you please! */
19
20  void *scs_loop(void *arg) {
21      (void) arg;
22      scs_do_step();
23      return NULL;
24  }
25
26  void *els_loop(void *arg) {
27      (void) arg;
28      light_do_step();
29      return NULL;
30  }
31
32
33  int main(int argc, char *argv[]) {
34      // TODO: get parameters from arguments in order to initialize the car
35      (void) argc; (void) argv;
36
37      init_system(leftHand, false, EU,false,false);
38
39      pthread_t els_thread;
40      pthread_t scs_thread;
41
42      pthread_create(&els_thread, NULL, els_loop, NULL);
43      pthread_create(&scs_thread, NULL, scs_loop, NULL);
44
45
46
47      return 0;
48  }
49
```

Details @ 2021-01-29 07:36:25

Violation Version	2021-01-29 07:36:25 (2021-01-29T07:36:25.178Z)
Id	abz-2020-case-study:SV12
Owners	NOBODY
Justification	
Location	main.c:43:19
Provider	axivion
Severity	required
Error Number	MisraC2012-17.7
Message	Return value of function discarded.
Entity	pthread_create()
Additional SLocs	/Library/Developer/CommandLineTools/SDKs/MacOSX11.1.sdk/usr/include/pthread/pthread.h:329:5

Rule @ 2021-01-29 07:36:25

Rule 17.7 The value returned by a function having non-*void* return type shall be *used*

Category: Required
Analysis: Decidable, Single Translation Unit
Applies to: C90, C99, C11

Rationale

It is possible to call a function without using the return value, which may be an error. If the return value of a function is intended not to be *used* explicitly, it should be cast to the *void* type. This has the effect of using the value without violating Rule 2.2.

Example

```
uint16_t func ( uint16_t para1 )
{
  return para1;
}

uint16_t x;
```

Static Analysis for Functional Requirements?

- Not an out-of-the-box experience
- Needs customization and analyses tailored to the use case
- But can be realized on top of existing analysis frameworks
- Could be done for requirements such as ‘’After engine start, there is no previous desired speed. The valid values for desired speed are from 1 km/h to 200 km/h.”

Static Analysis - Results

- Very helpful to ensure MISRA C2012 is adhered to
- Very helpful to uncover programming errors and security issues
  - Null pointer dereferences
  - Division by zero
  - Bad practices
  - …
- Many things can be checked out of the box without tweaking
- Can sometimes be used for functional requirements, but that involves customizations

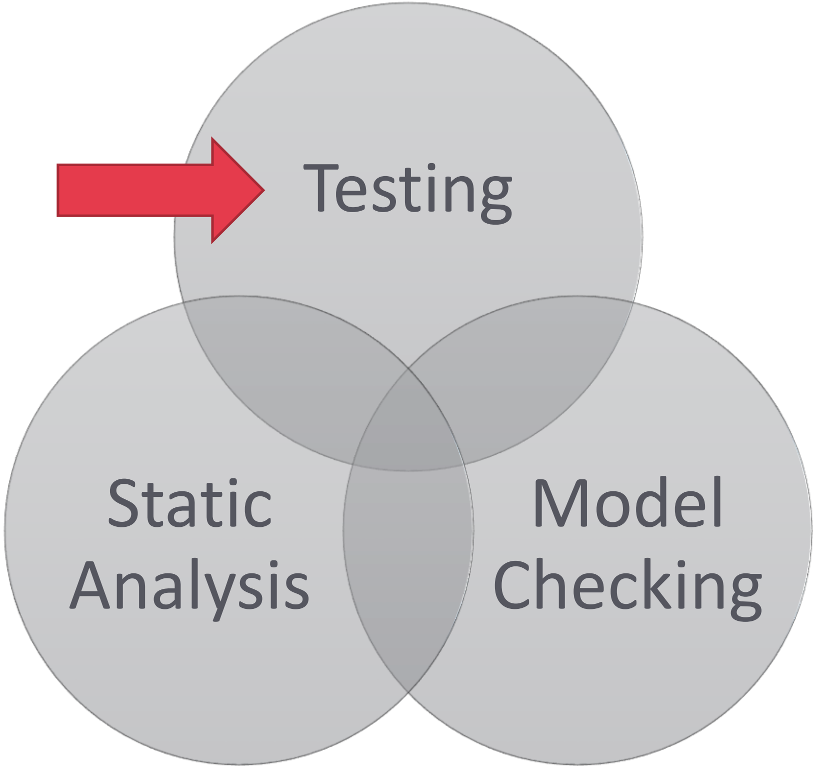
Formal Proof and Deductive Verification

- Broadly speaking: show correctness of system with mathematical rigor
- Translate system into some mathematical / logical representation
- Use (semi-)automatic or manual proof techniques to proof properties
- Very high effort and skill needed

Formal Proof is Possible!

Image by Pline under CC BY-SA 3.0

Combining Techniques



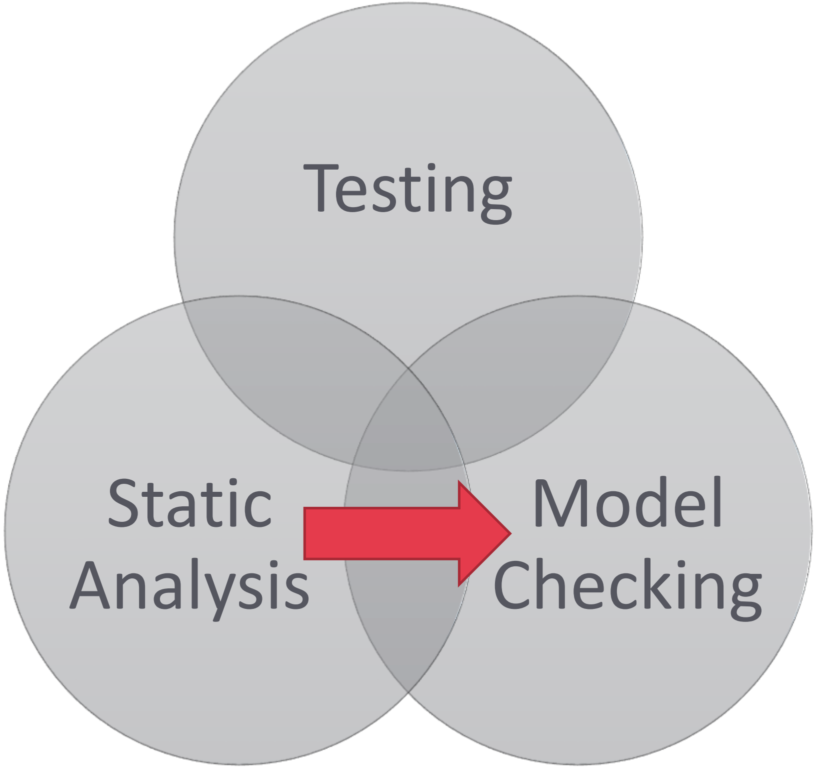
Testing

Static Analysis

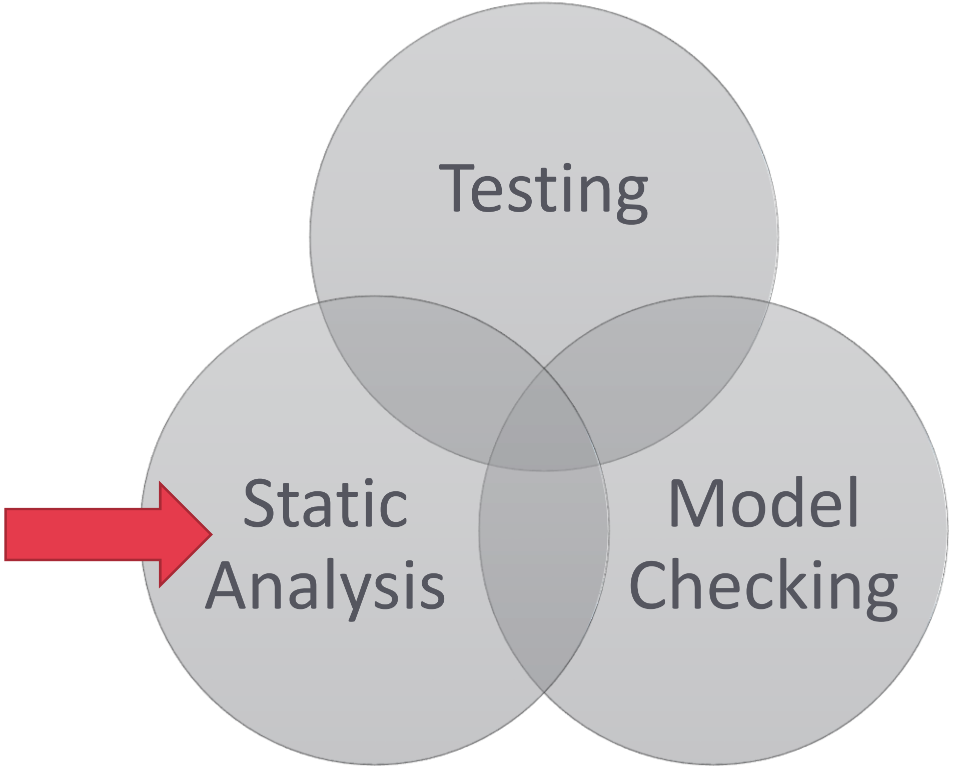
Model Checking

Combining Techniques

axivion
stopping software erosion



Combining Techniques

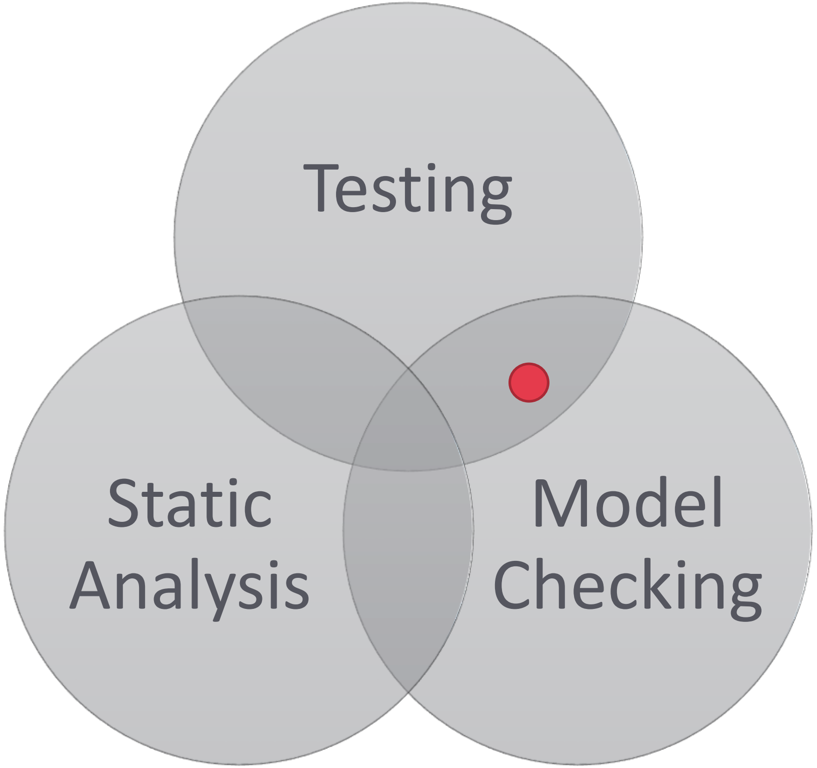


Testing

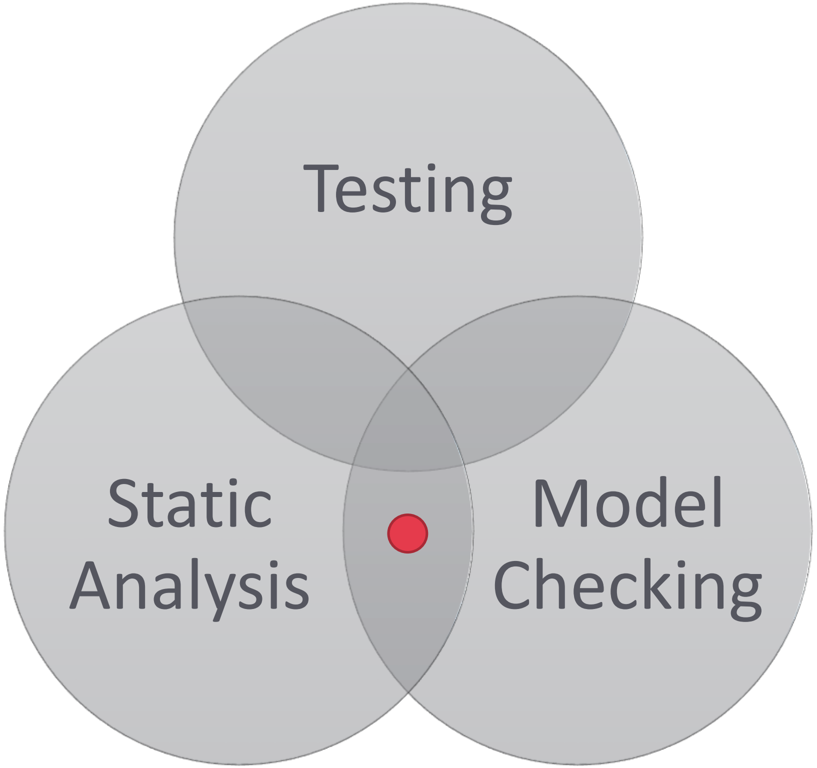
Static Analysis

Model Checking

Combining Techniques

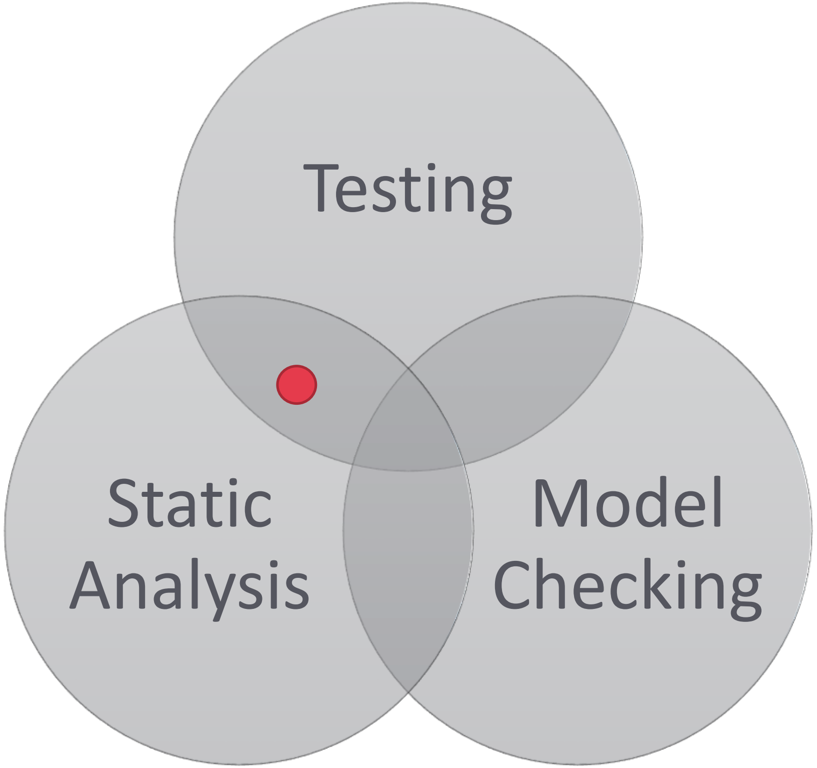


Combining Techniques

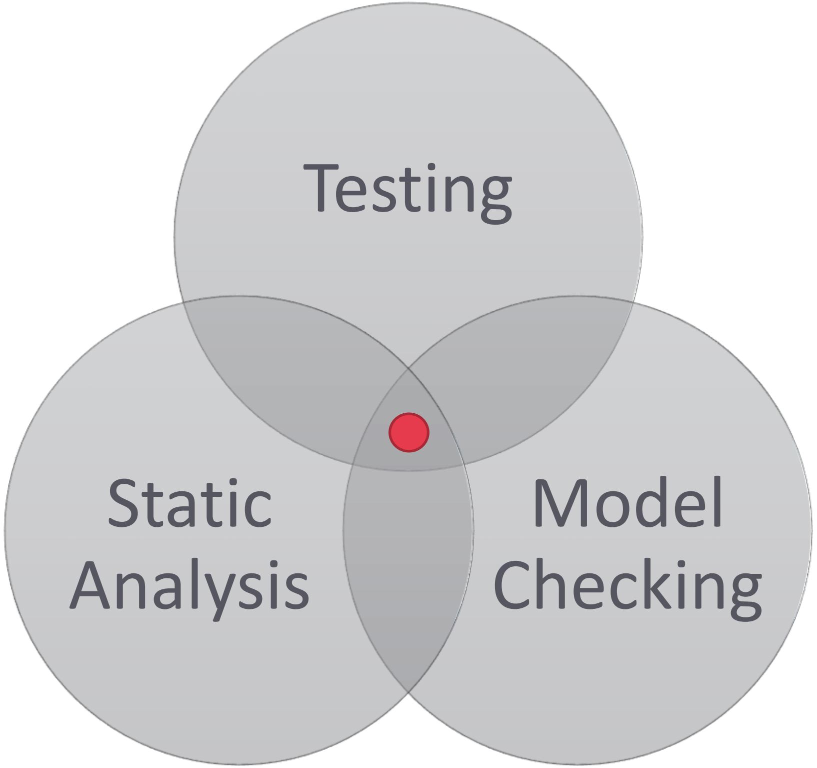


Combining Techniques

axivion
stopping software erosion



Combining Techniques



Conclusions

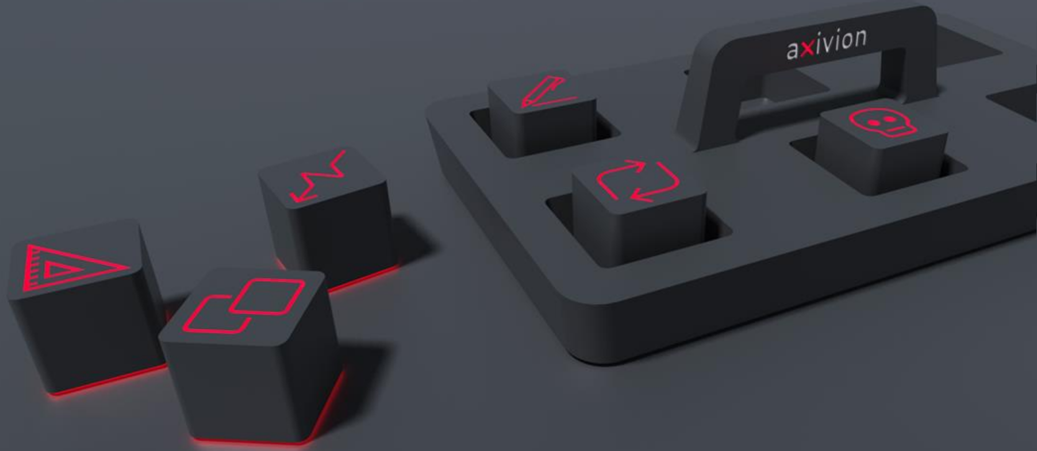
- Use whatever you can to ensure safety and security
- Techniques alone are useful …
- … but upcoming combinations might be even better.

Stay tuned for what we come up with!

Thank you for your attention!

Contact:
Sebastian Krings
Axivion GmbH
Nobelstraße 15
70569 Stuttgart

Tel: +49 711 6204378-78
E-Mail: krings@axivion.com
www.axivion.com



stopping software erosion