ESD Electronic Systems Design
Synthesis Verification Testing Power Communication

# Model-Driven Design and Validation of Embedded Software


*Giuseppe Di Guglielmo*[†]
*Masahiro Fujita*[†]
*Cristina Marconcini*[‡]
*Andreas Foltinek*[•]

***Luigi Di Guglielmo***[*]
*Franco Fummi*[*]
*Graziano Pravadelli*[*]

[†] VDEC
*University of Tokyo*
*Japan*

[‡] STM Products srl
*Italy*

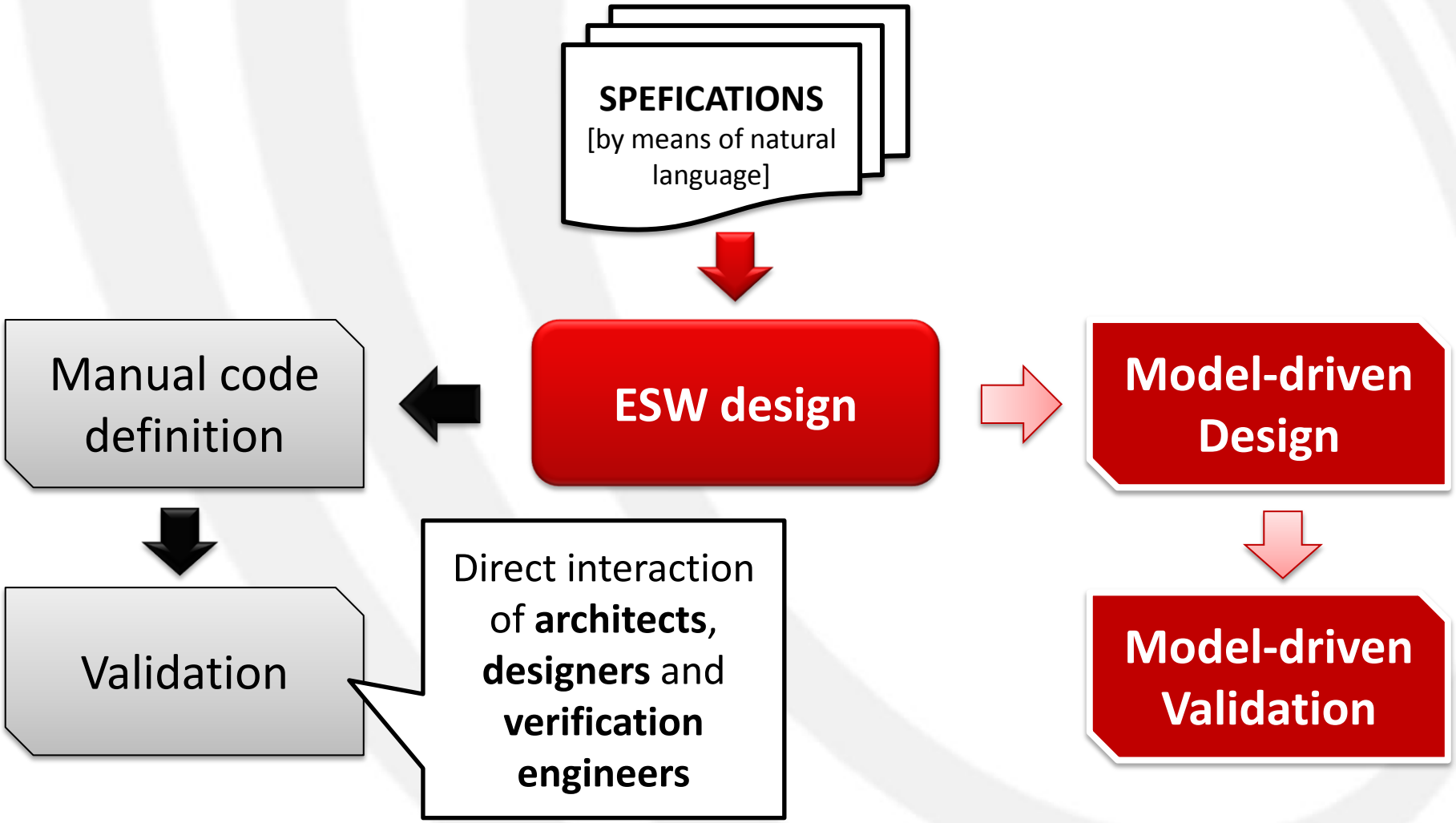
[•] IMACS GmbH
*Germany*

[*] Dep. Computer Science
*University of Verona*
*Italy*

# Outline

- Motivations
- State of Art
- Design and Validation framework
  - rad*CASE*
  - rad*CHECK*
- Comparisons
- Conclusions

# Motivations



# State of Art (I)

- **Model-driven Design** (*MDD*)
  - Simplifies the modeling by raising the abstraction level of the description
    - Use of generic models for architecture and behaviors
    - Generic in the choice of coding languages
    - Generic in the choice of execution platforms
  - Advantages:
    - Most suitable solution to develop complex systems
  - Drawbacks:
    - Need of a validation framework

# State of Art (II)

- **Model-driven Validation** (*MDV*)
  - Aims at guaranteeing that the described system models correctly the requirements
  - It may combine
    - Automatic generation of the testcases
    - Assertion-based verification

# State of Art (III)

- **Automatic generation of the testcases**
  - Concrete execution
    - Pseudo-random techniques, genetic and probabilistic algorithms
  - Symbolic execution
    - As a “better” concrete execution
    - Exhaustive reasoning along paths
  - Advantages:
    - Avoid the time consuming process of manual test writing
    - Eases the adoption of dynamic assertion-based approaches
  - Drawbacks:
    - Corner cases may escape if the generation algorithm is not effective

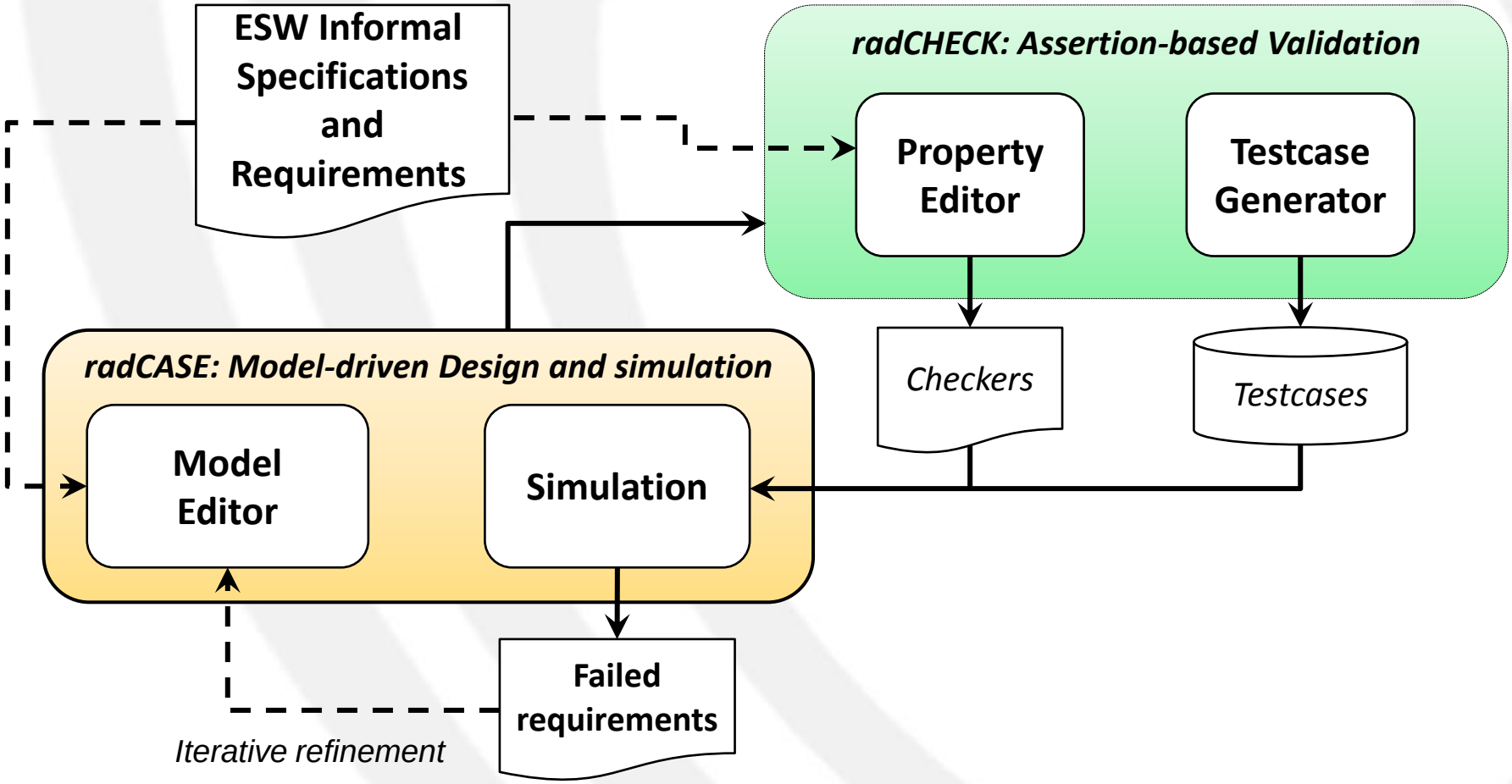
# State of Art (IV)

- **Assertion-based Verification** (*ABV*)
  - Speeds up the detection and the debug of the system errors by using properties
    - Properties are the formalization of the system requirements
    - A property falsification notifies an error into the model
  - Advantages:
    - Enables the detection of corner cases errors
  - Drawbacks:
    - Complexity of formal languages requires qualified engineers for the property definition process

# Goals

- Comprehensive framework for the design and validation of ESW that exceeds the limitations of current Model-driven approaches providing the user with
  - **rad*CASE***
    - an effective UML-modeling and code-generation environment
  - **rad*CHECK***
    - an effective validation environment that includes guided property definition and automatic testcases generation

# Overview


ESW Informal Specifications and Requirements
radCHECK: Assertion-based Validation
Property Editor
Testcase Generator
Checkers
Testcases
radCASE: Model-driven Design and simulation
Model Editor
Simulation
Failed requirements
Iterative refinement
Manual activity
Automatic activity

# rad*CASE*

- rad*CASE*: model-driven design and simulation
  - The ESW model is described by means of UML diagrams and C-code
  - The code implementation is automatically generated
    - Support for different target platforms
  - A graphical simulator allows the manually validation of the generated code

# rad*CHECK*

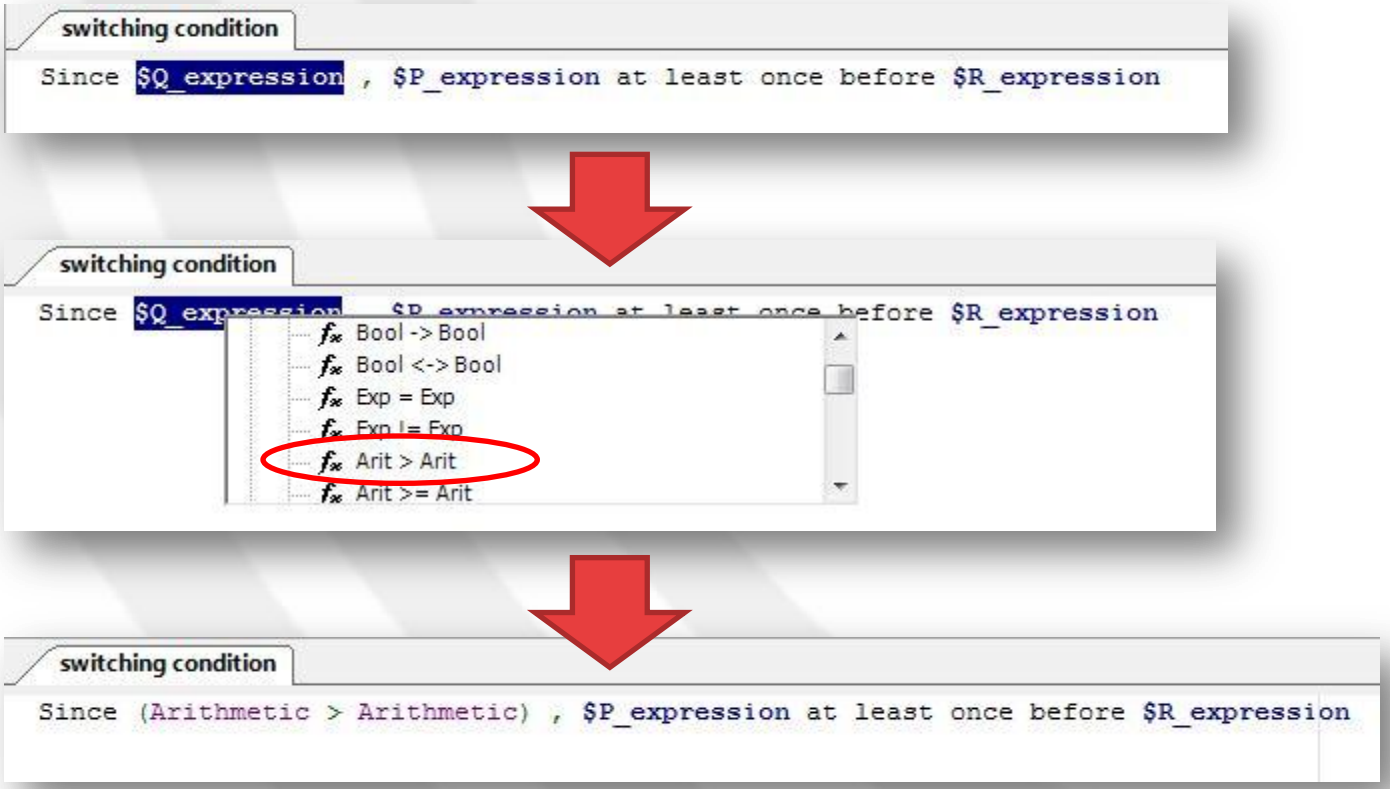
- rad*CHECK*: assertion-based validation
  - Property definition
  - Checker generation
  - Testcases generation
- Automate the validation phase

# rad*CHECK*: Property definition (I)

- A property editor guides the formalization of the informal requirements into properties
  - Property templates
  - PSL grammar rules
- Property Specification Language (PSL)
  - PSL semantics that allows to formalize complex temporal behaviors in a precise and concise manner

# radCHECK: Property definition (II)

- Example of use of a parametric template



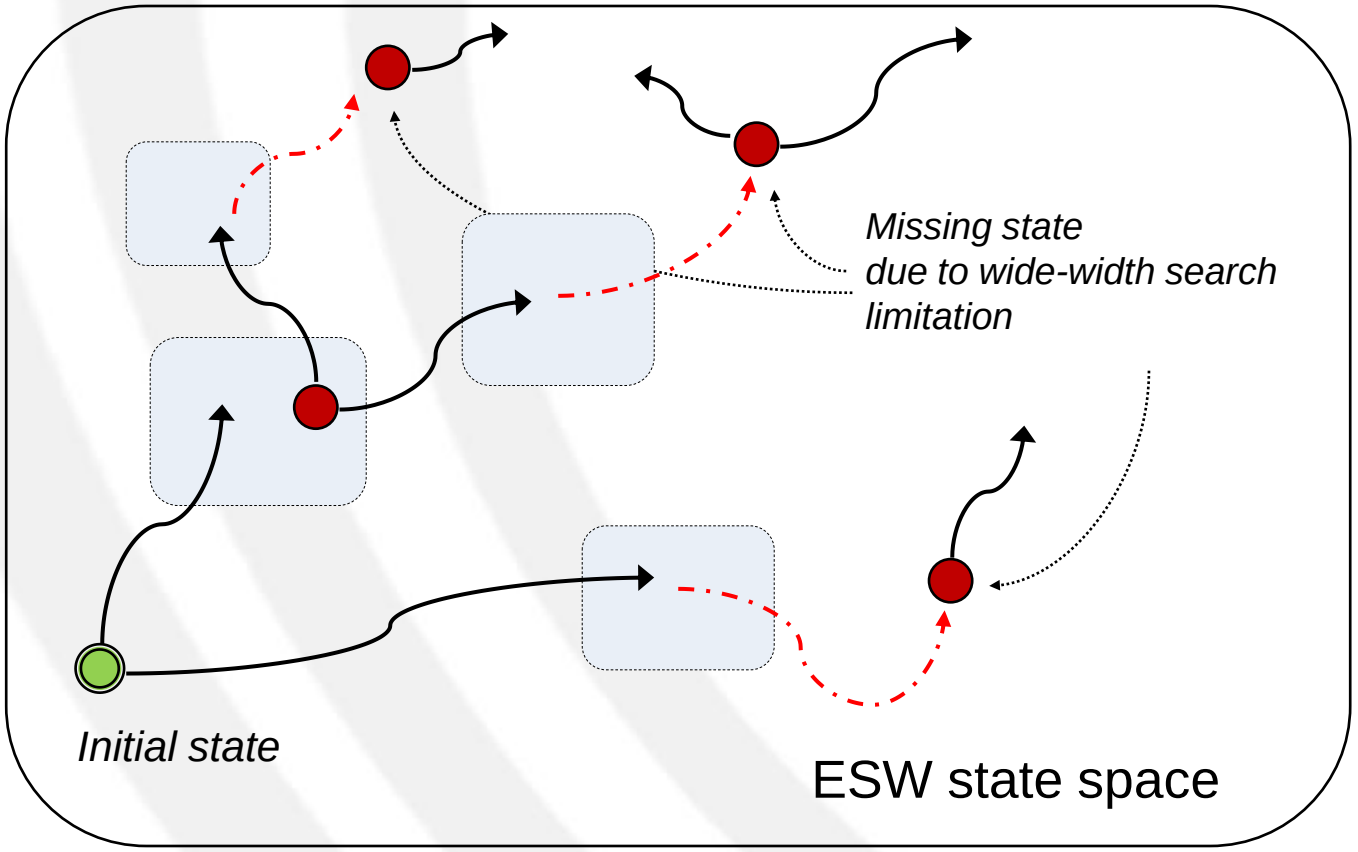
# rad*CHECK*: Checker Generator (I)

- A checker implements the behavior described by a property
  - It monitors the evolution of a set of variables verifying that their values do not violate the specified property
- A checker consists of C-code executed in parallel with ESW implementation

# rad*CHECK*: Checker Generator (II)

- The checker generation
  1. Parsing the property for generating a tree structure that represents the dependences between the property sub-formulas
  2. Mapping each sub-formula into an internal Boolean variable
  3. Mapping each temporal and logical operator into a function that implements the operator meaning
  4. Combining these functions according to the tree structure to model the whole property behavior

# radCHECK: Testcases Generation




Corner-case state  Concrete execution  Symbolic execution  Guided Symbolic execution

# Tools Comparison

- rad*CASE*
  - Comparison with some of the most-used UML model-driven tools in the field of ESW design
- rad*CHECK*
  - Property editor
    - Novel idea
  - Checker generator
    - IBM FoCs
  - Automatic testcases generator for ESW

# Model-driven Design Tools Comparison

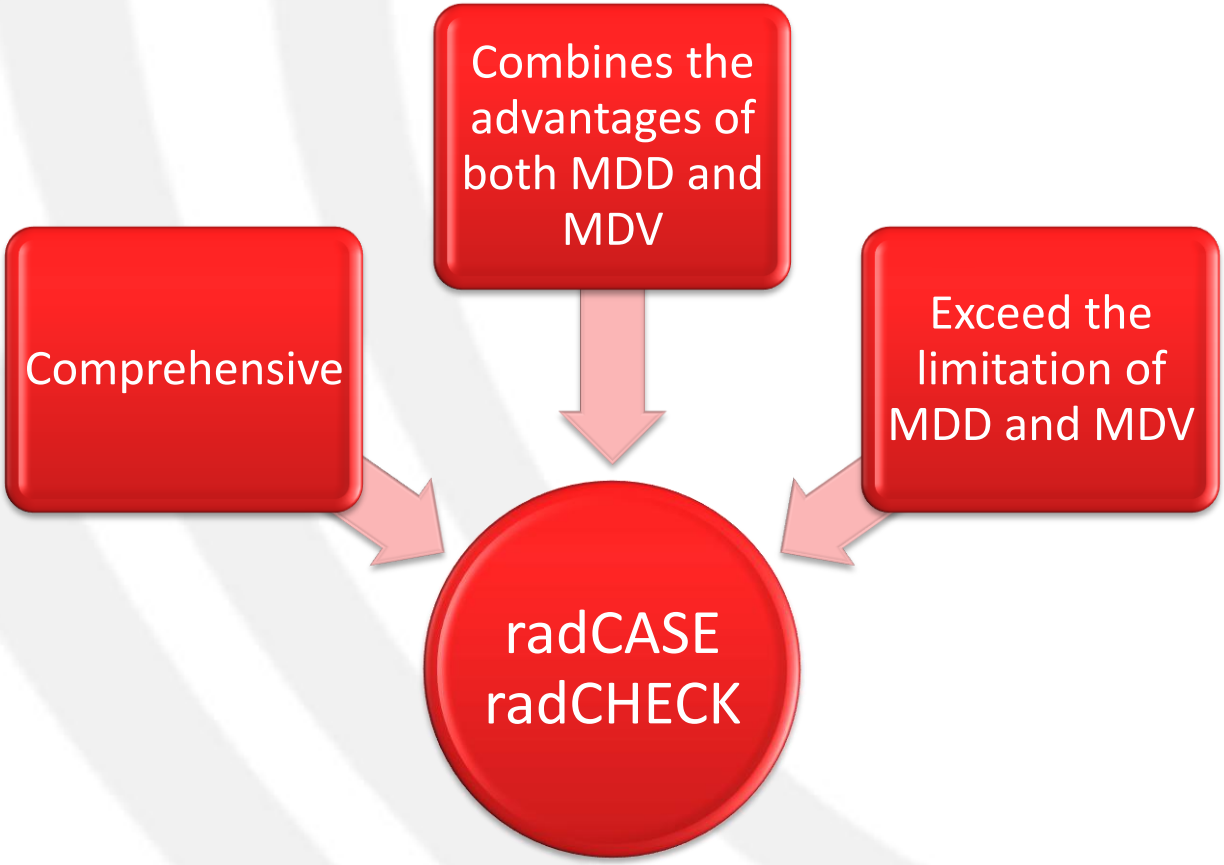
| Category | Feature | radCASE | ARTiSAN | Enterprise Architect | Rapsody | VisualSTATE |
|---|---|---|---|---|---|---|
| Require-ments | Use Case | Y | Y | Y | Y | - |
| ESW structure | Class Diag. | Y | Y | Y | Y | - |
| | Object Diag. | Y | Y | Y | Y | - |
| | Component Diag. | Y | Y | Y | Y | - |
| ESW behavior | State machines | Y | Y | Y | Y | Y |
| | Sequence Diag. | Y | Y | Y | Y | - |
| | Activity Diag. | Y | Y | Y | Y | - |
| | FBDs | Y | - | - | - | - |
| | C/C++ | Y | Y | Y | Y | - |
| Validation | Simulator | Y | Y | - | Y | - |
| GUI | HMI Editor | Y | - | - | - | - |

# Checker Generators Comparison

| PSL operator | CG | IBM FoCs |
|---|---|---|
| next, next_a, next_e | Y | Y |
| next!, next_a!, next_e! | Y | - |
| next_event | Y | Y |
| next_event! | Y | - |
| next_event_a | Y | - |
| next_event_a! | Y | - |
| next_event_e | Y | - |
| next_event_e! | Y | - |
| before | Y | Y |
| before! | Y | - |
| until | Y | Y |

| PSL operator | CG | IBM FoCs |
|---|---|---|
| until! | Y | Y |
| eventually! | Y | Y |
| always | Y | Y |
| logical operators | Y | Y |
| SERE suffix implication | Y | Y |
| SERE consecutive rep. | Y | Y |
| SERE non-consecutive rep. | Y | Y |
| SERE goto repetition | Y | Y |
| SERE or | Y* | Y* |
| SERE length match. And | Y* | Y* |
| SERE non-len. match. And | Y* | Y* |

# Conclusions



# On-line References

- **rad*CASE***
  http://www.stm-case.com

- **rad*CHECK***
  http://www.verificationsuite.com