*21st ACM International Conference on Hybrid Systems: Computation and Control (HSCC'18)*

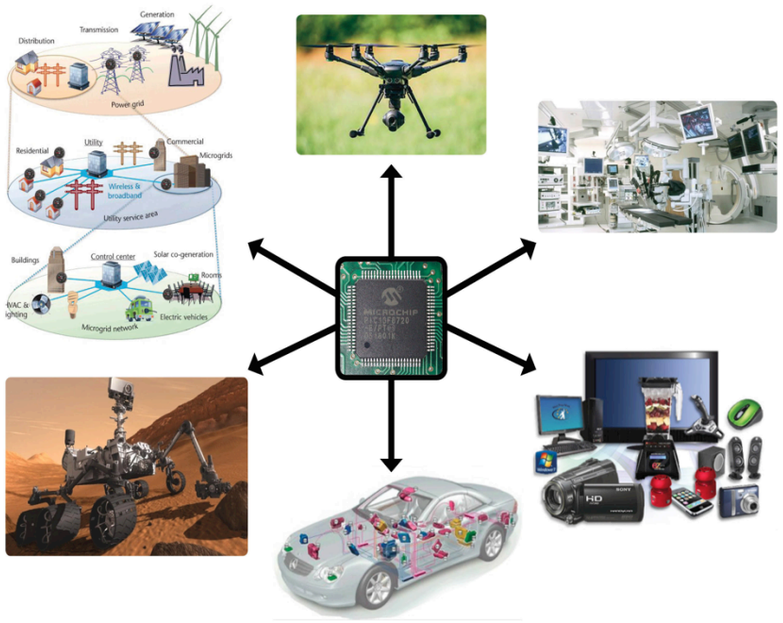
# *DSValidator: An Automated Counterexample Reproducibility Tool for Digital Systems*

*Joint work with Lennon Chaves, Iury Bessa, and Daniel Kroening*

*Lucas Cordeiro*
*University of Oxford*
*lucas.cordeiro@cs.ox.ac.uk*

# Establish Trust in Verification Results

Specification

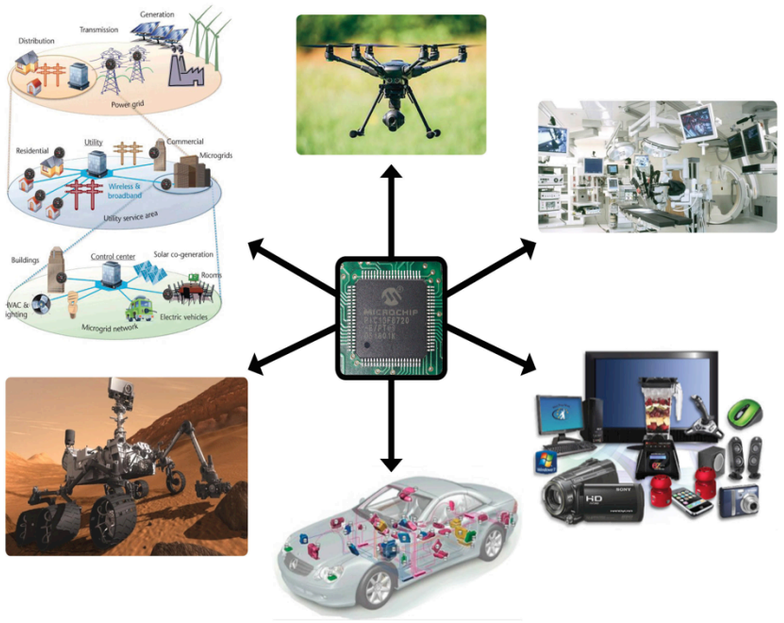



Implementation

Digital Controller and Filter

# Establish Trust in Verification Results

Specification



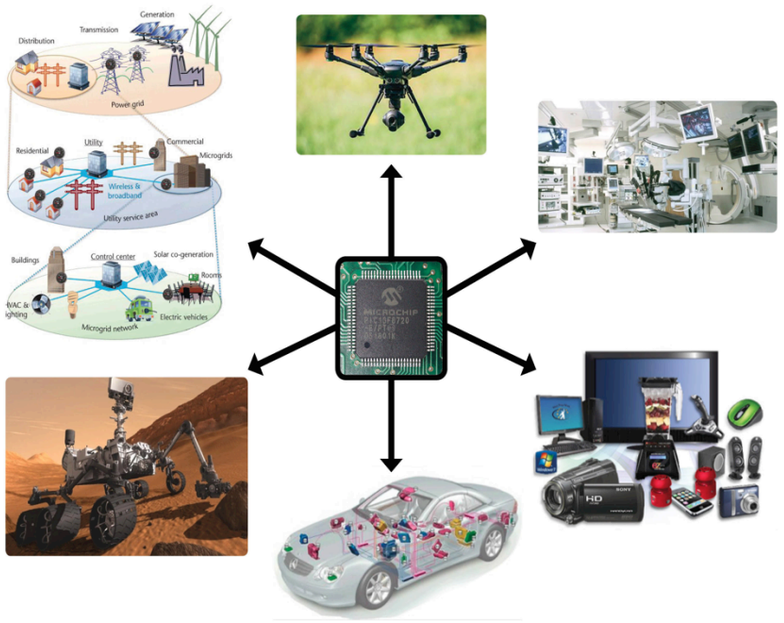

Digital System Verifiers

Implementation

Digital Controller and Filter

# Establish Trust in Verification Results

Specification




Implementation

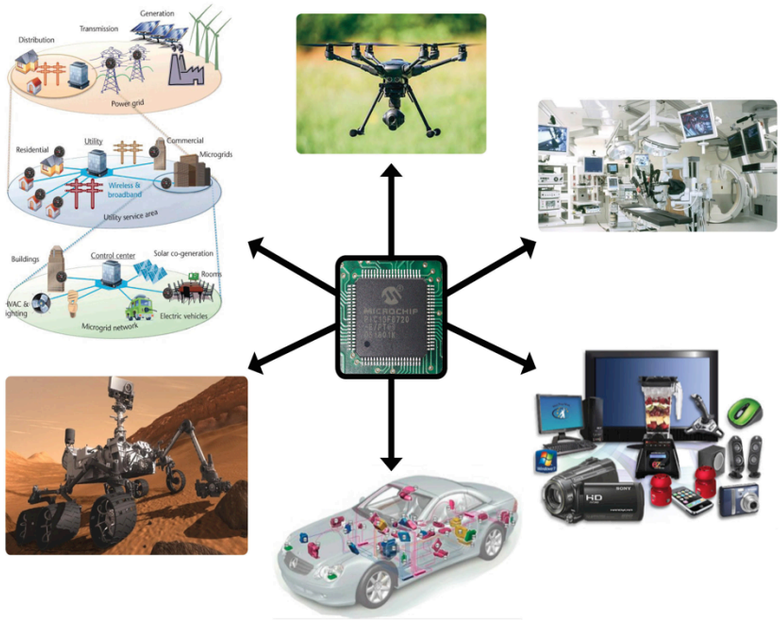
Verification Successful

Digital System Verifiers

Digital Controller and Filter

# Establish Trust in Verification Results

Specification




Digital Controller and Filter

Verification Successful

Digital System Verifiers

DSValidator

Counter-example

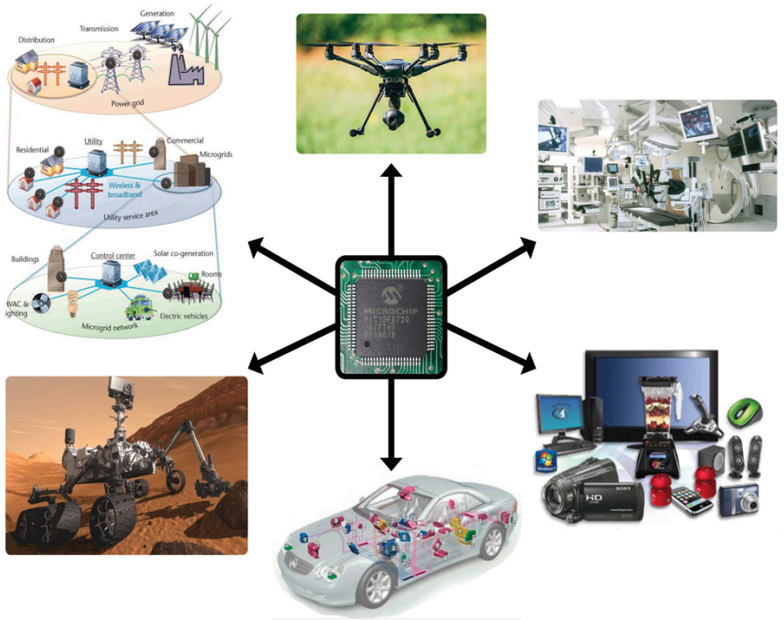
CE Reproducible

Fix the implementation

Implementation

# Establish Trust in Verification Results

Specification




Digital Controller and Filter

Verification Successful

Digital System Verifiers

CE Irreproducible

Incorrect result

DSValidator

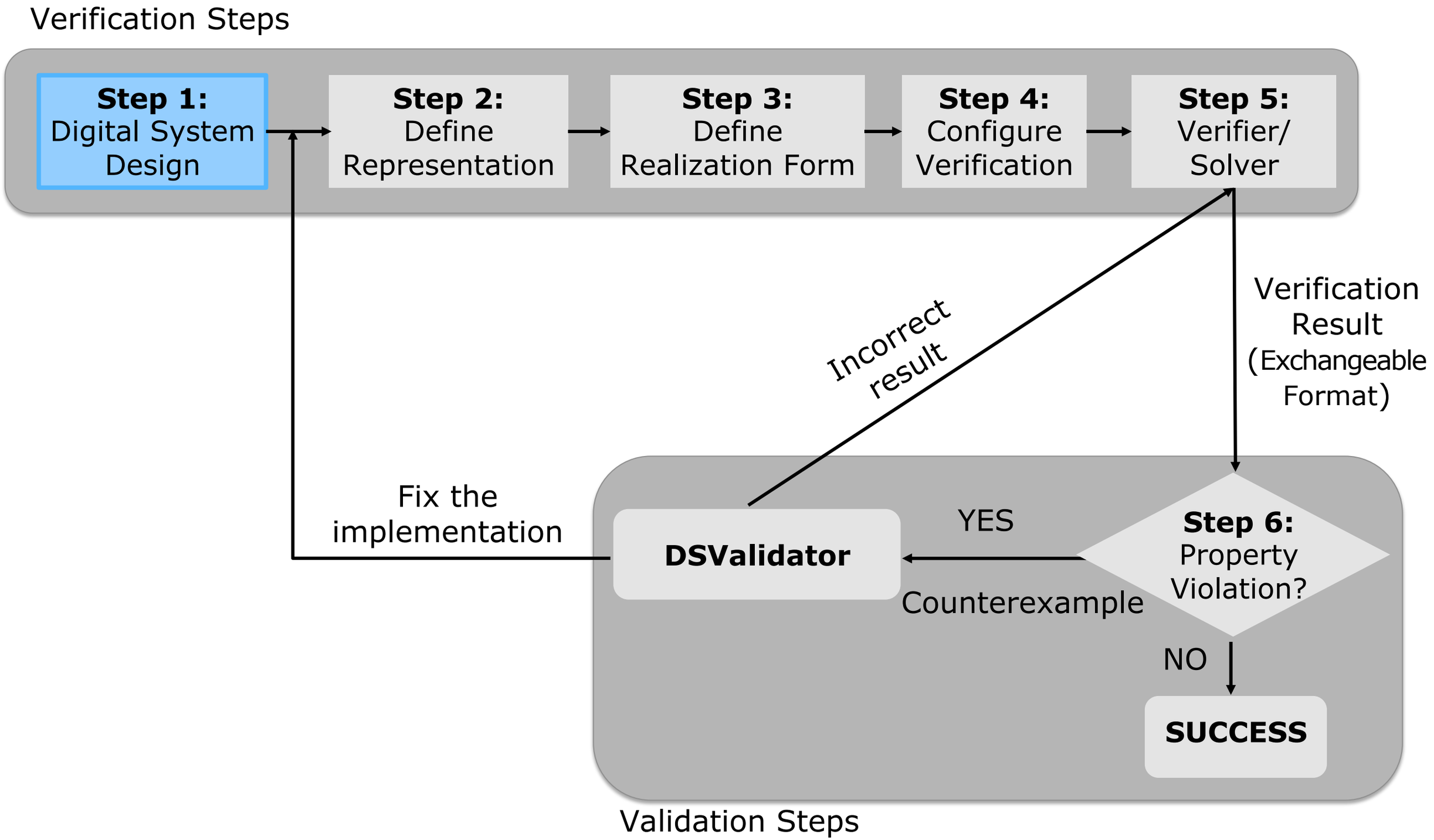
Counter-example

CE Reproducible

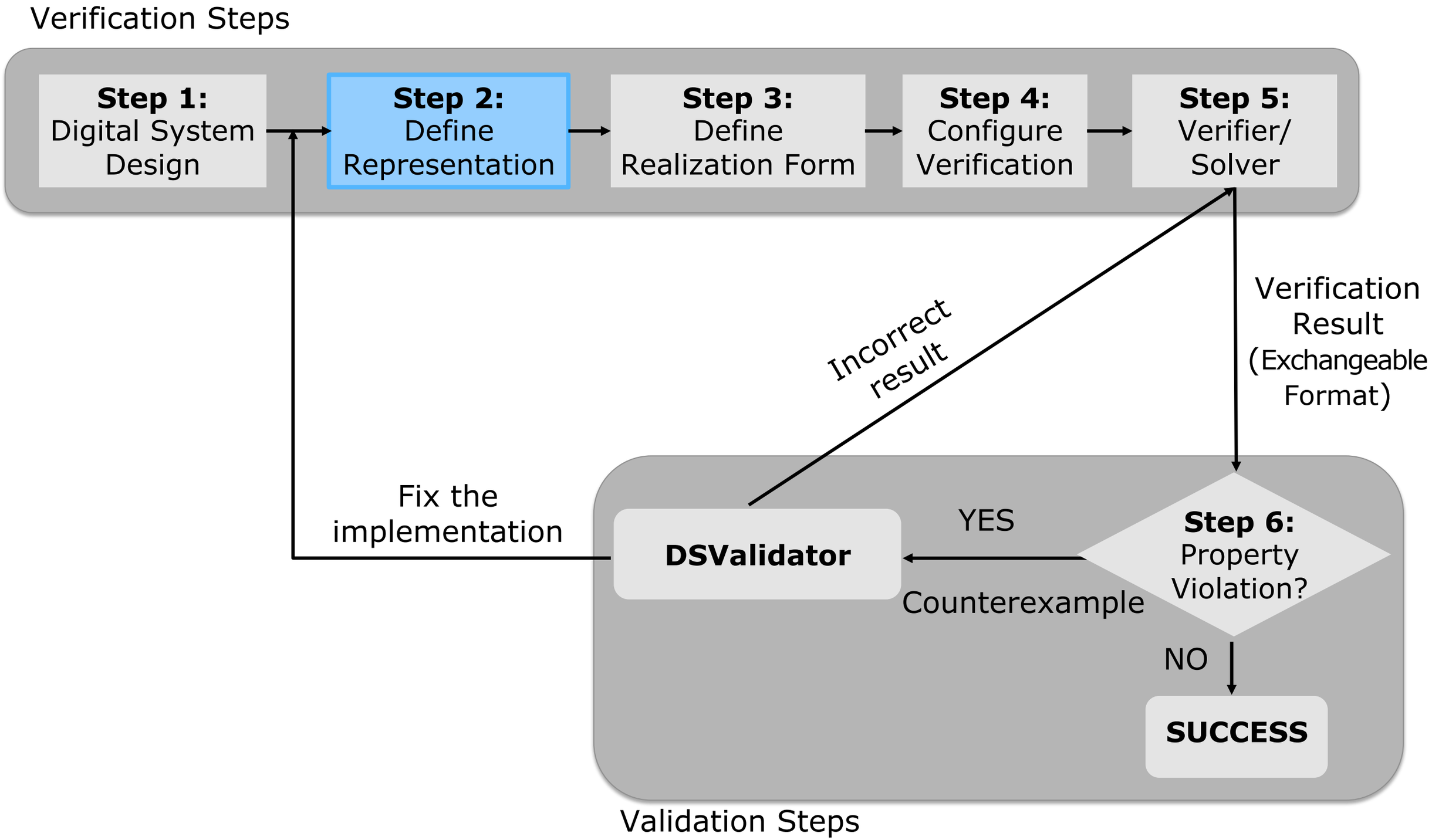
Fix the implementation

Implementation

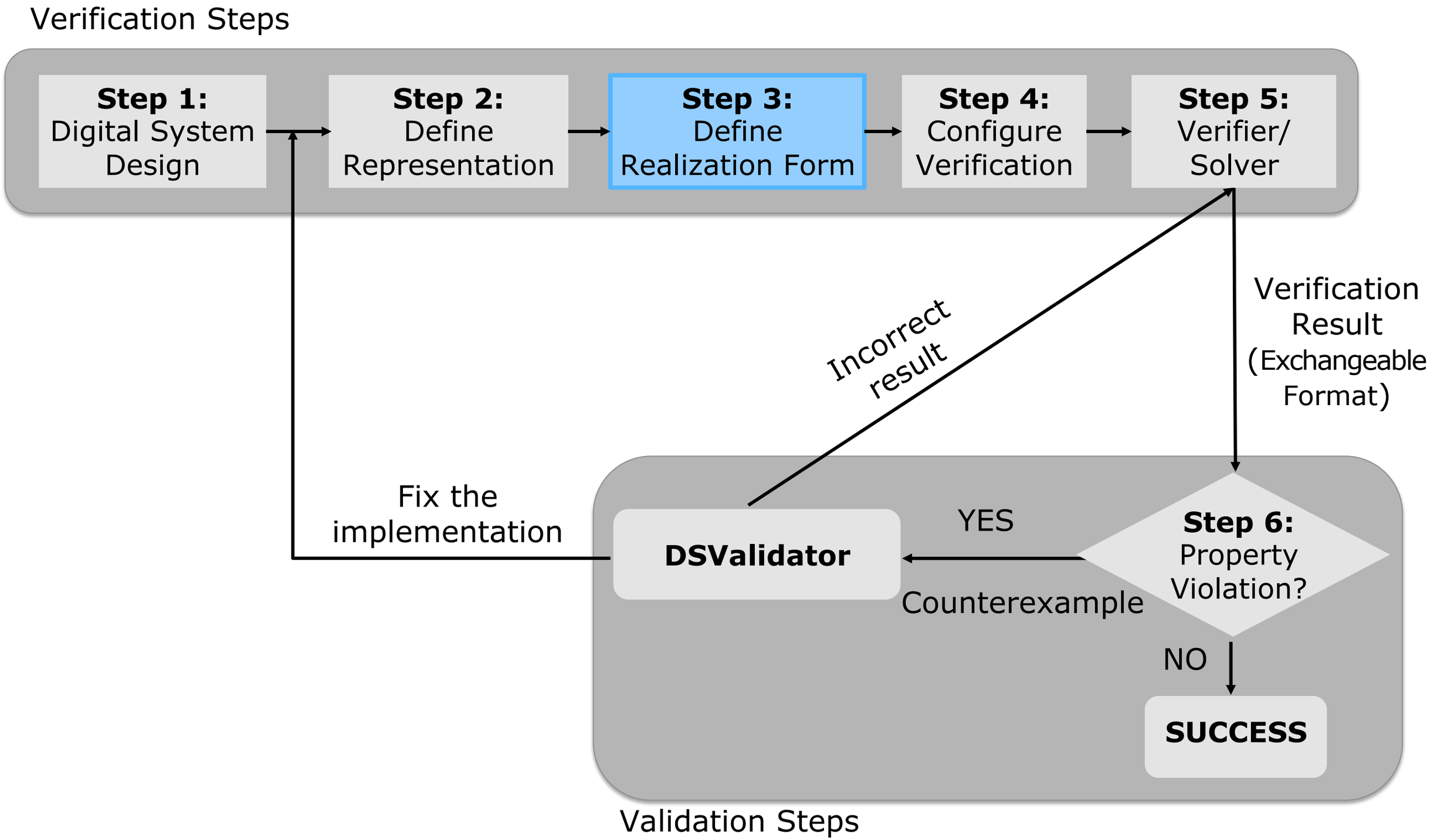
# Verification & Validation Methodology



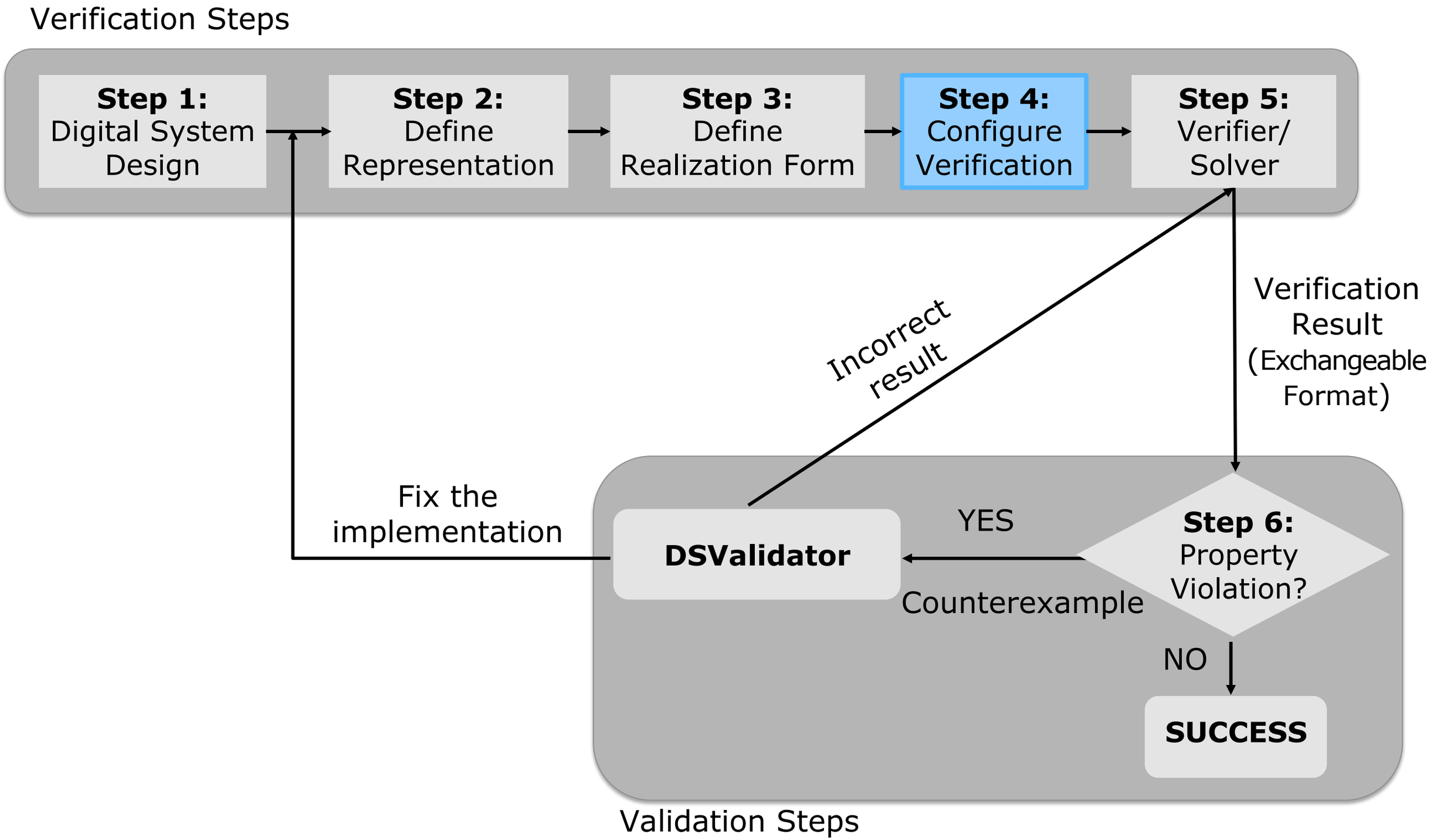
# Verification & Validation Methodology



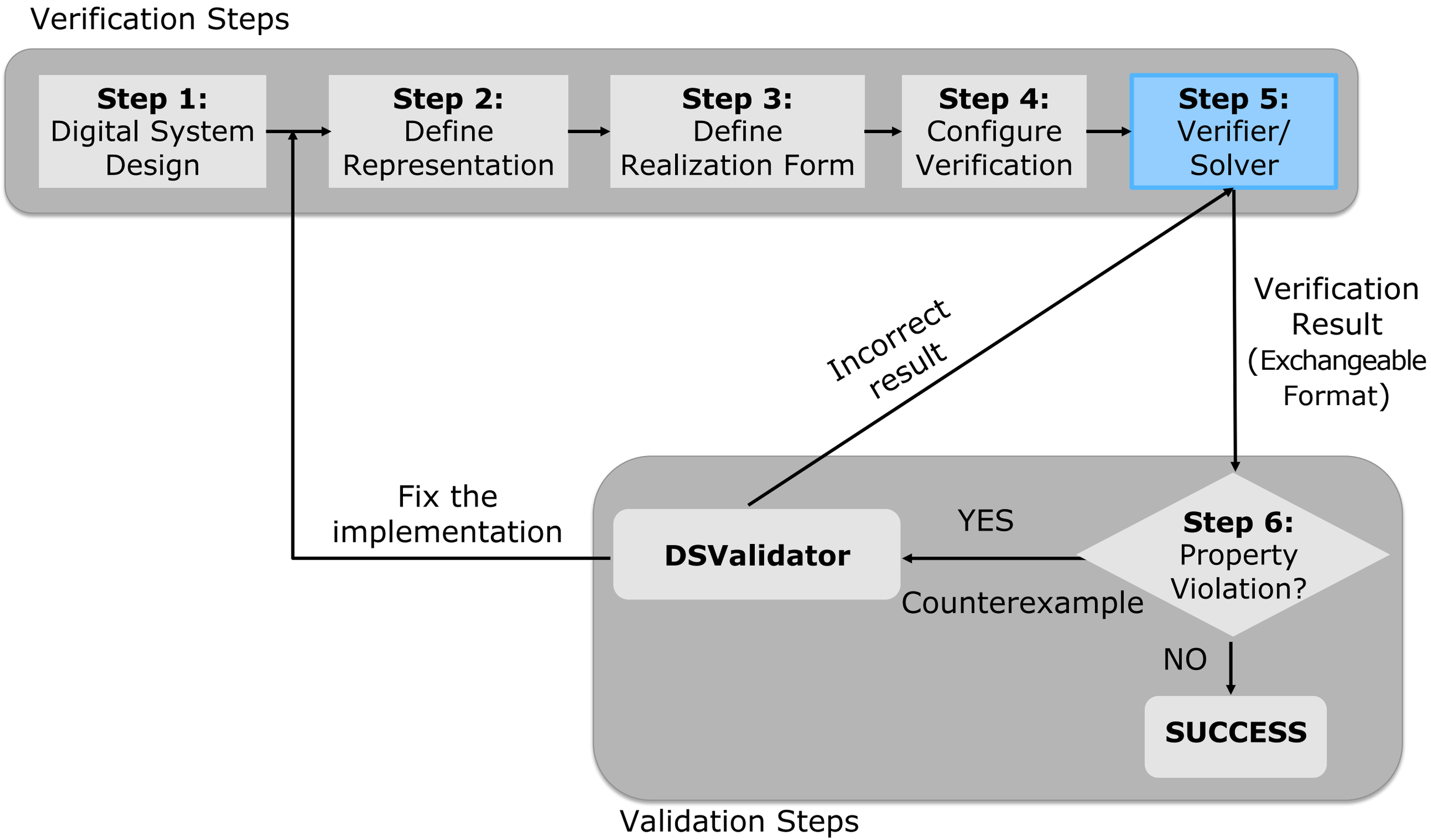
# Verification & Validation Methodology



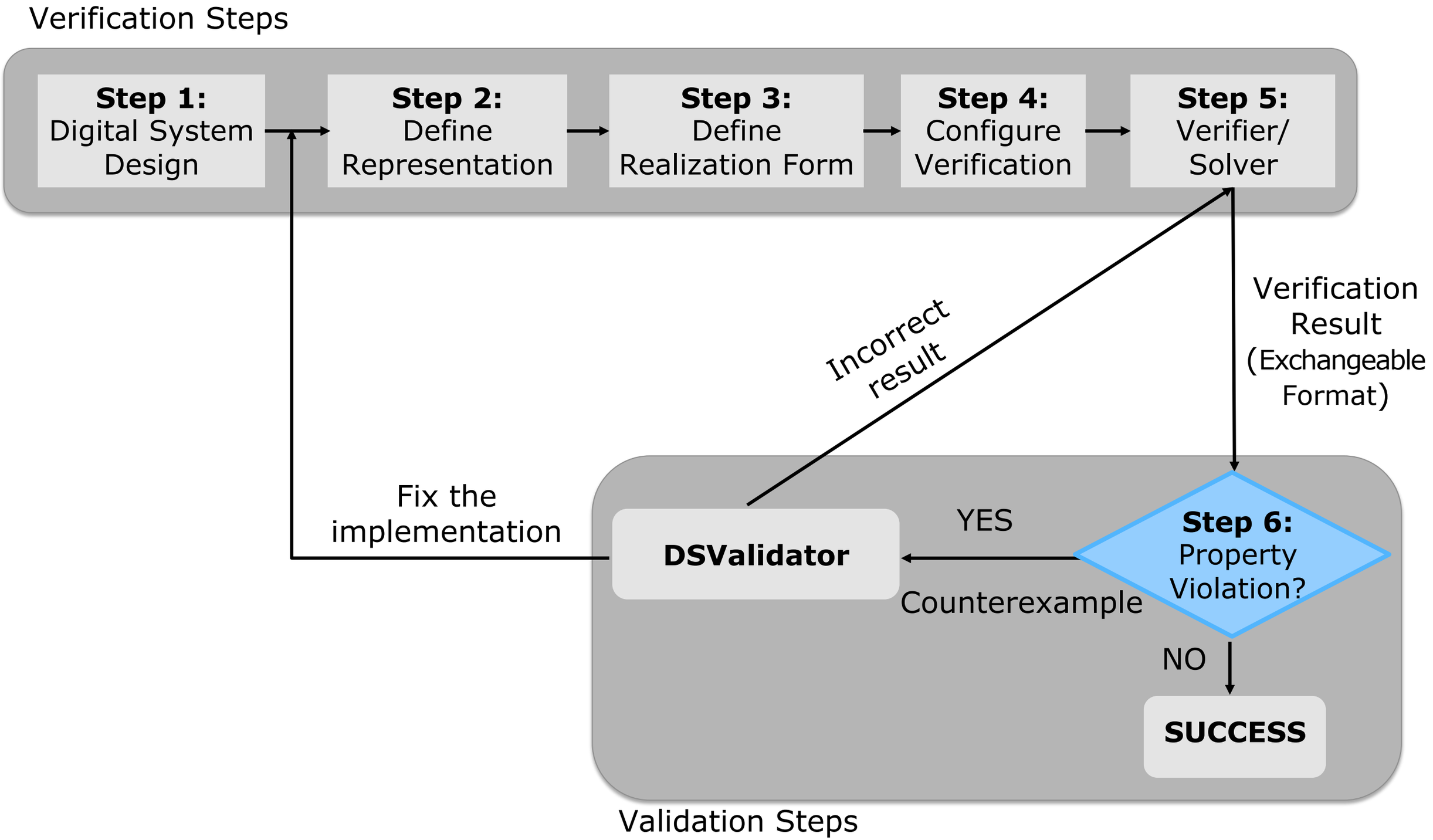
# Verification & Validation Methodology



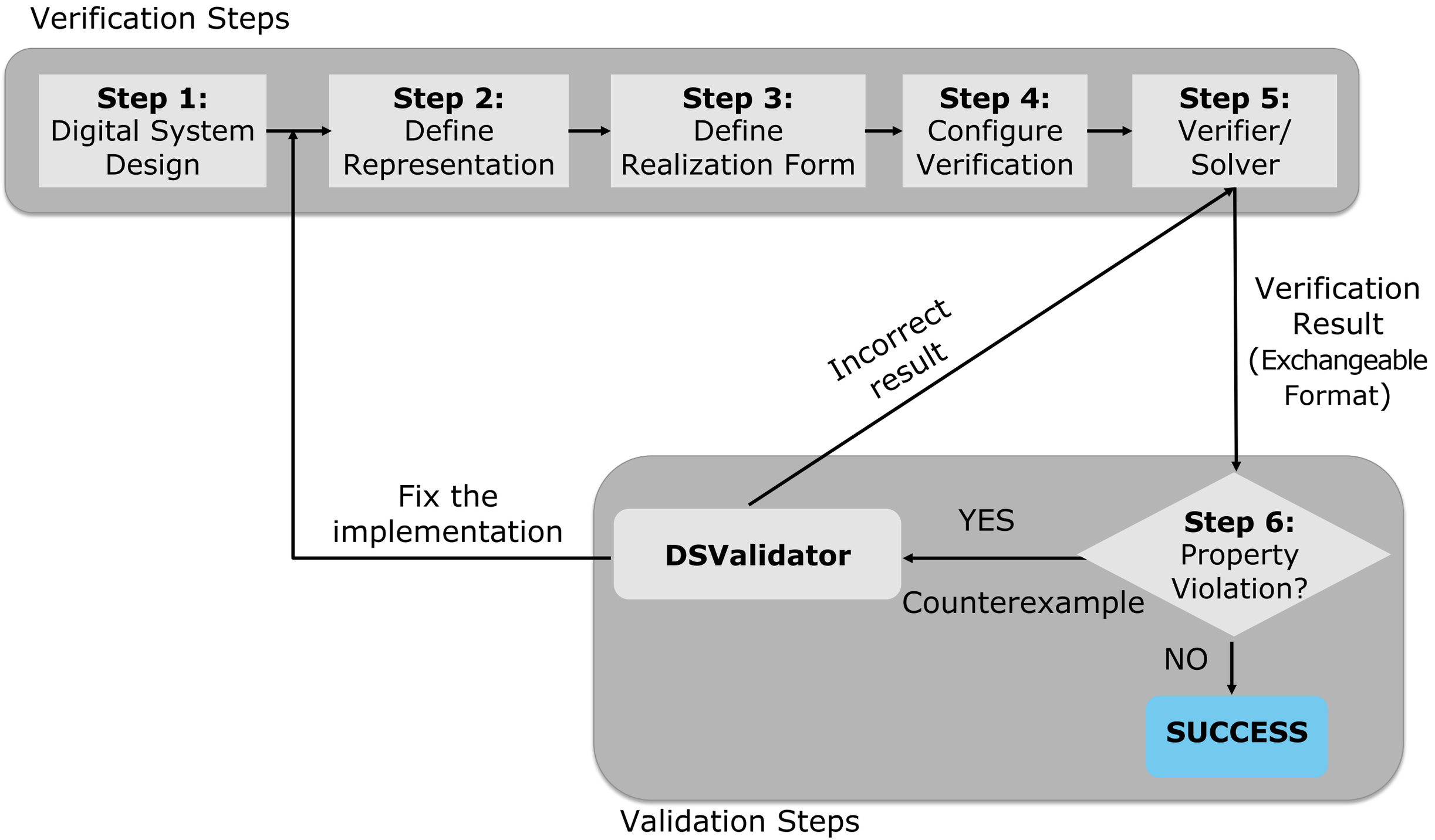
# Verification & Validation Methodology



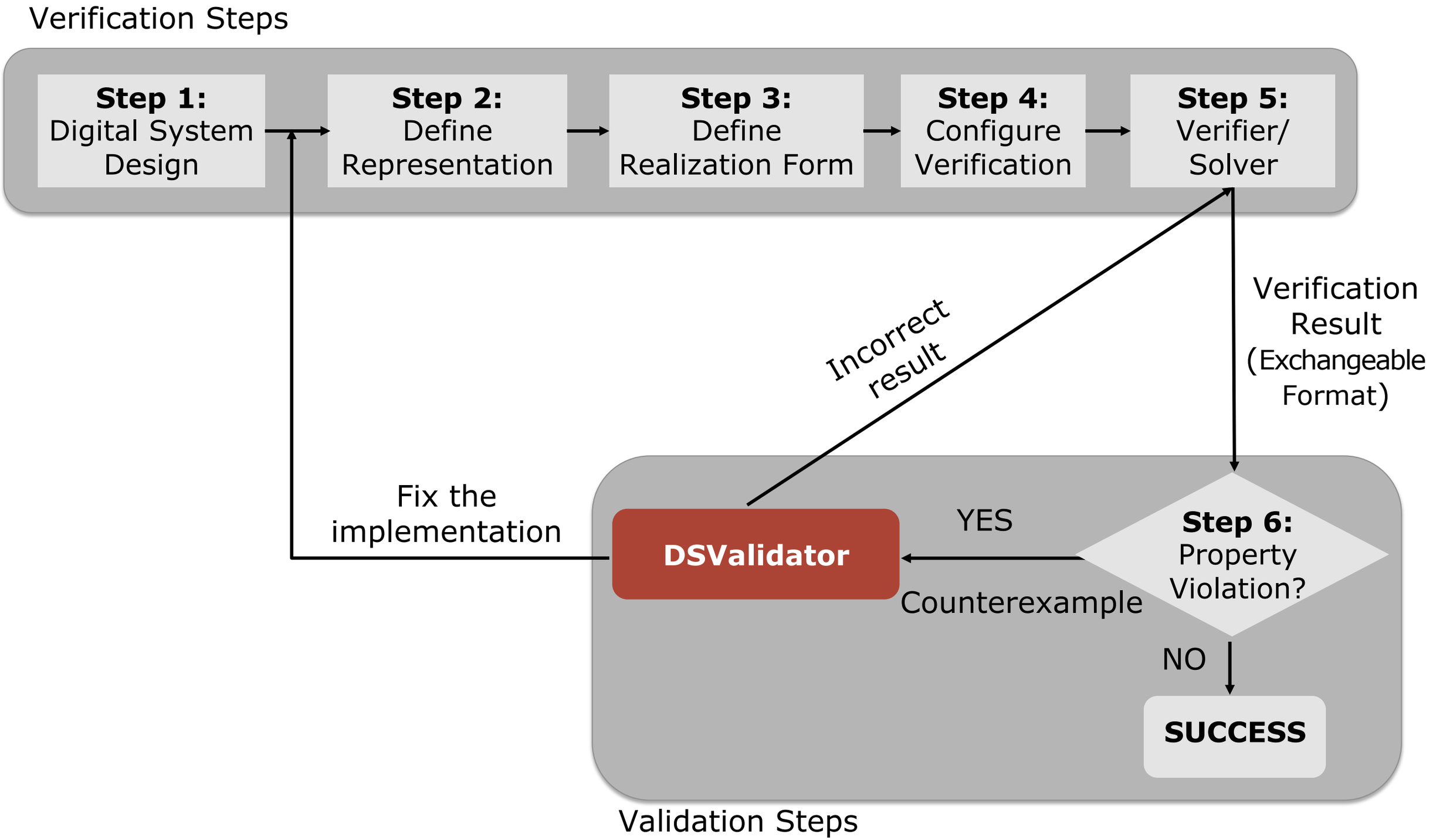
# Verification & Validation Methodology



# Verification & Validation Methodology



# Verification & Validation Methodology



# Objectives

**Establish trust in verification results for digital systems**

# Objectives

**Establish trust in verification results for digital systems**

- Propose a format to represent the counterexamples that can be used by any verifier

# Objectives

**Establish trust in verification results for digital systems**

- Propose a format to represent the counterexamples that can be used by any verifier
- Reproduce counterexamples that refute properties related to limit cycle, overflow, stability and minimum-phase

# Objectives

**Establish trust in verification results for digital systems**

- Propose a format to represent the counterexamples that can be used by any verifier
- Reproduce counterexamples that refute properties related to limit cycle, overflow, stability and minimum-phase
- Validate a set of intricate counterexamples for digital controllers used in a real quadrotor attitude system

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1}
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = OVERFLOW
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.02
Implementation = <10,6>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamic_Range = {-1, 1}
Inputs = { -1, -0.75, 0.0, -0.5, 0.0, 0.25, 1, -0.5,
0.078125, 0.6875 }
Outputs = { -2002, 2498.5, -1000.0, -1.0, 1000.0, -499.5,
2002, -5001, 6156, -4936.125 }
```

# DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = OVERFLOW
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.02
Implementation = <10,6>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamic_Range = {-1, 1}
Inputs = { -1, -0.75, 0.0, -0.5, 0.0, 0.25, 1, -0.5,
0.078125, 0.6875 }
Outputs = { -2002, 2498.5, -1000.0, -1.0, 1000.0, -499.5,
2002, -5001, 6156, -4936.125 }
```

# DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by  a **transfer function**:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}}$$

# DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by a **transfer function**:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}}$$

- Computes
  - **finite-word lengths effects** over the $a_k$ and $b_k$ coefficients
  - **roots of a polynomial** for stability and minimum-phase

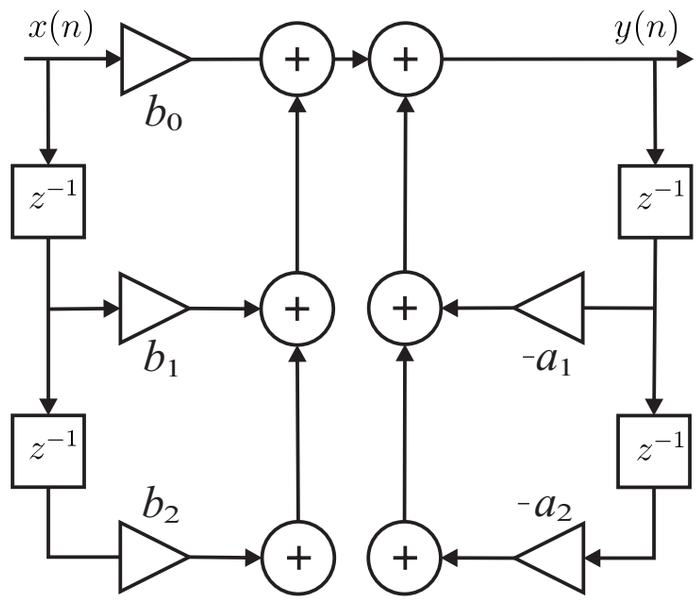
# DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by a **transfer function**:

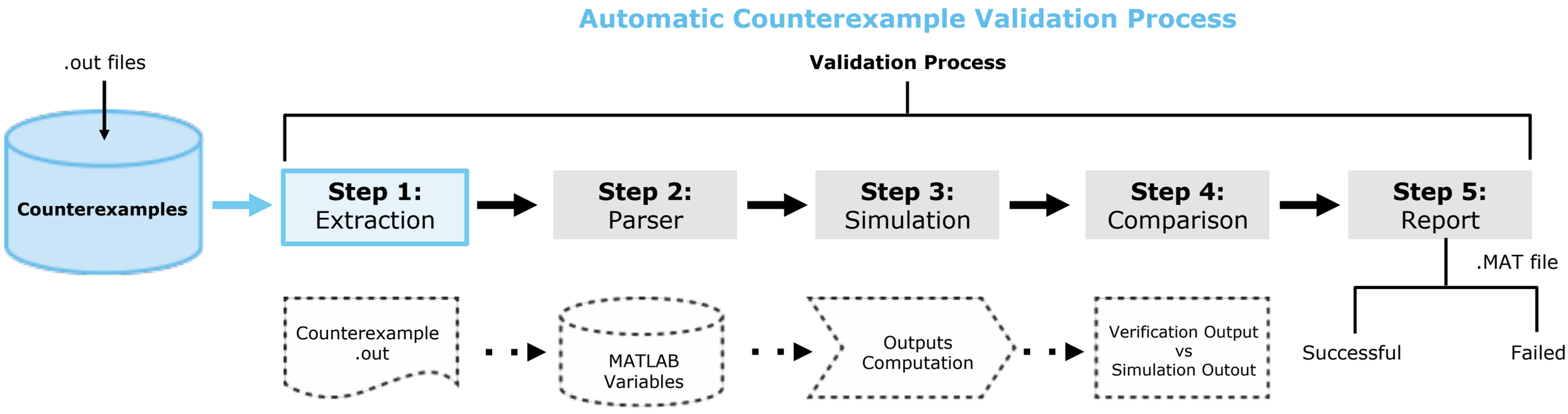
$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}}$$

- Computes
  - **finite-word lengths effects** over the $a_k$ and $b_k$ coefficients
  - **roots of a polynomial** for stability and minimum-phase
- Unrolls the system for a given **realization form**
  - overflow, granular LCO, overflow LCO

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

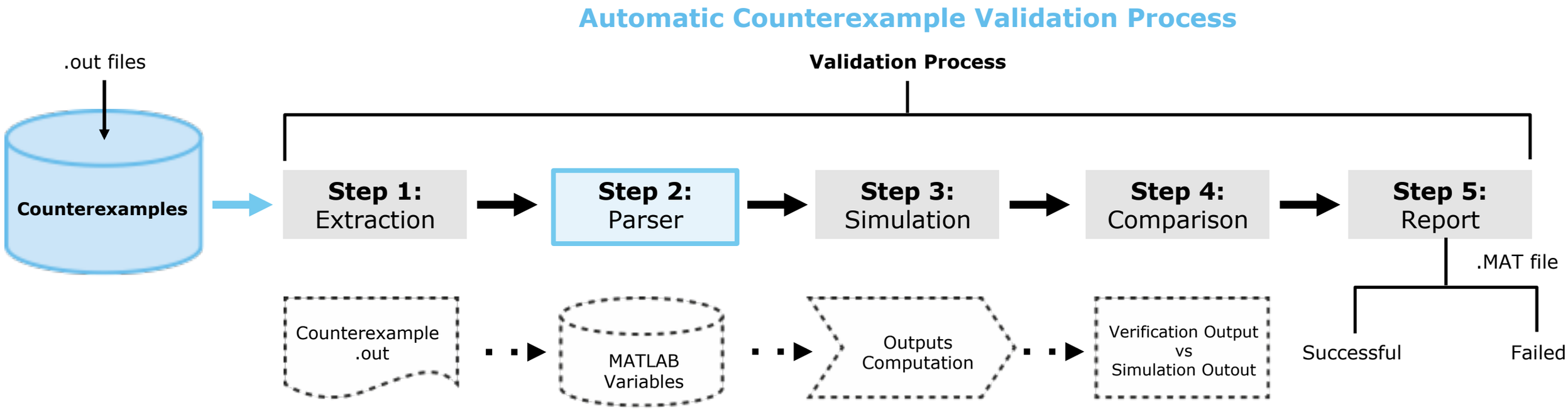


# DSValidator Validation Process



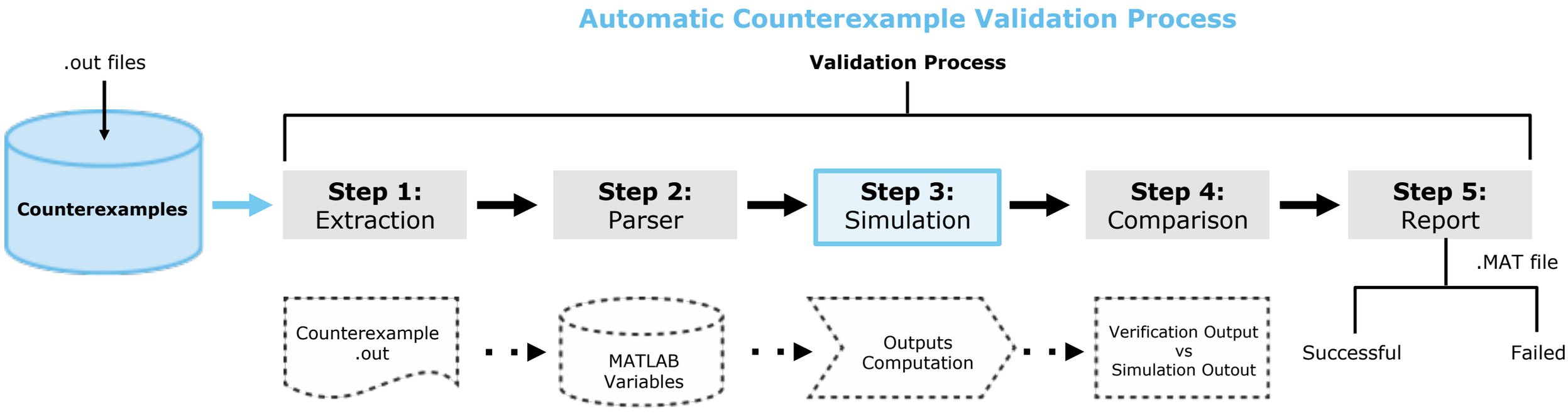

- **Extraction**
  - obtains the counterexample from the verifier

# DSValidator Validation Process



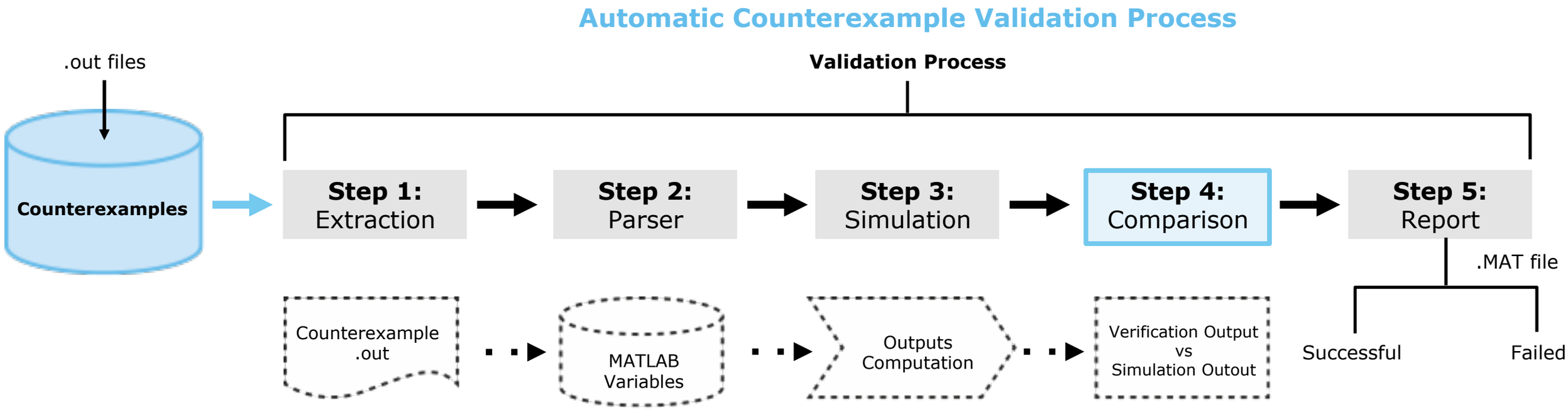

- **Extraction**
  - obtains the counterexample from the verifier
- **Parser**
  - converts all counterexample attributes into variables

# DSValidator Validation Process



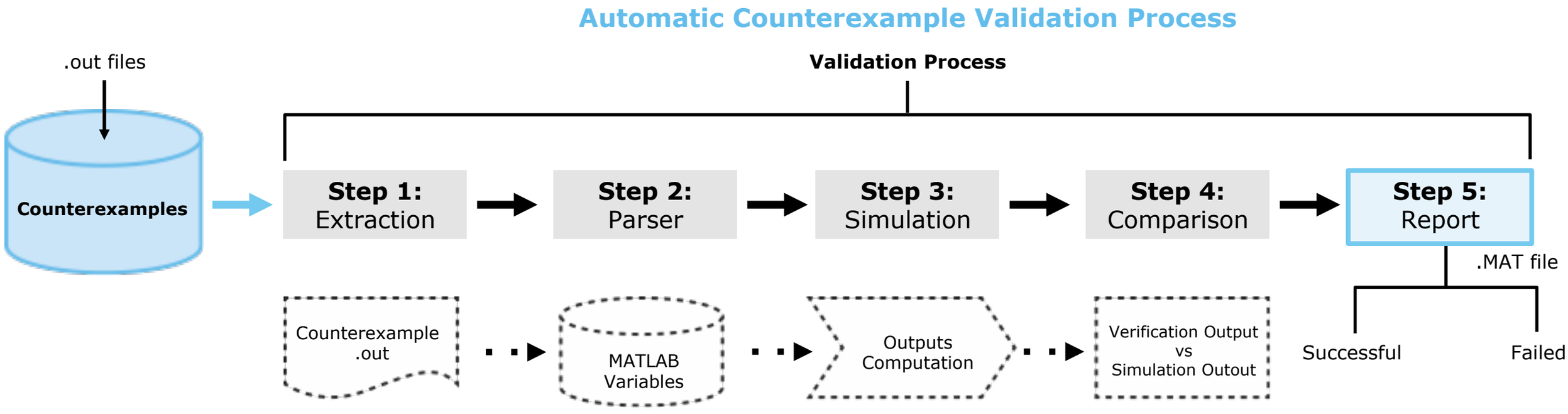

- **Extraction**
  - obtains the counterexample from the verifier
- **Parser**
  - converts all counterexample attributes into variables
- **Simulation**
  - simulates the counterexample (violation) for the failed property

# DSValidator Validation Process




- **Extraction**
  - obtains the counterexample from the verifier
- **Parser**
  - converts all counterexample attributes into variables
- **Simulation**
  - simulates the counterexample (violation) for the failed property
- **Comparison**
  - checks MATLAB simulation vs verifier output

# DSValidator Validation Process




- **Extraction**
  - obtains the counterexample from the verifier
- **Parser**
  - converts all counterexample attributes into variables
- **Simulation**
  - simulates the counterexample (violation) for the failed property
- **Comparison**
  - checks MATLAB simulation vs verifier output
- **Report**
  - stores the counterexample in a .MAT file and reports its reproducibility

# DSValidator Features

- **Validation Functions**
  - reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)

# DSValidator Features

- **Validation Functions**
  - reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)
- **Properties**
  - checks and validates overflow, limit-cycle, stability and minimum-phase

# DSValidator Features

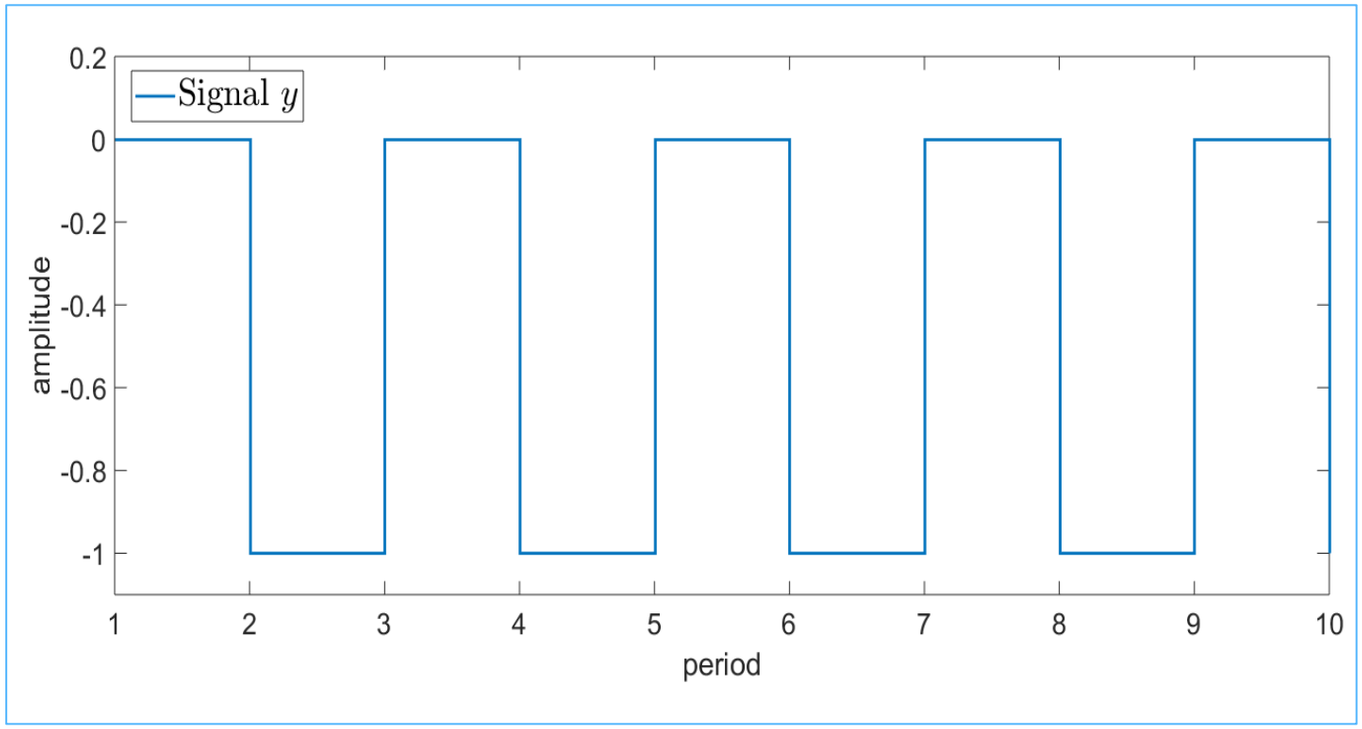
- **Validation Functions**
  - reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)
- **Properties**
  - checks and validates overflow, limit-cycle, stability and minimum-phase
- **Realization**
  - reproduces realization forms to validate overflow and limit-cycle (for direct and delta forms)

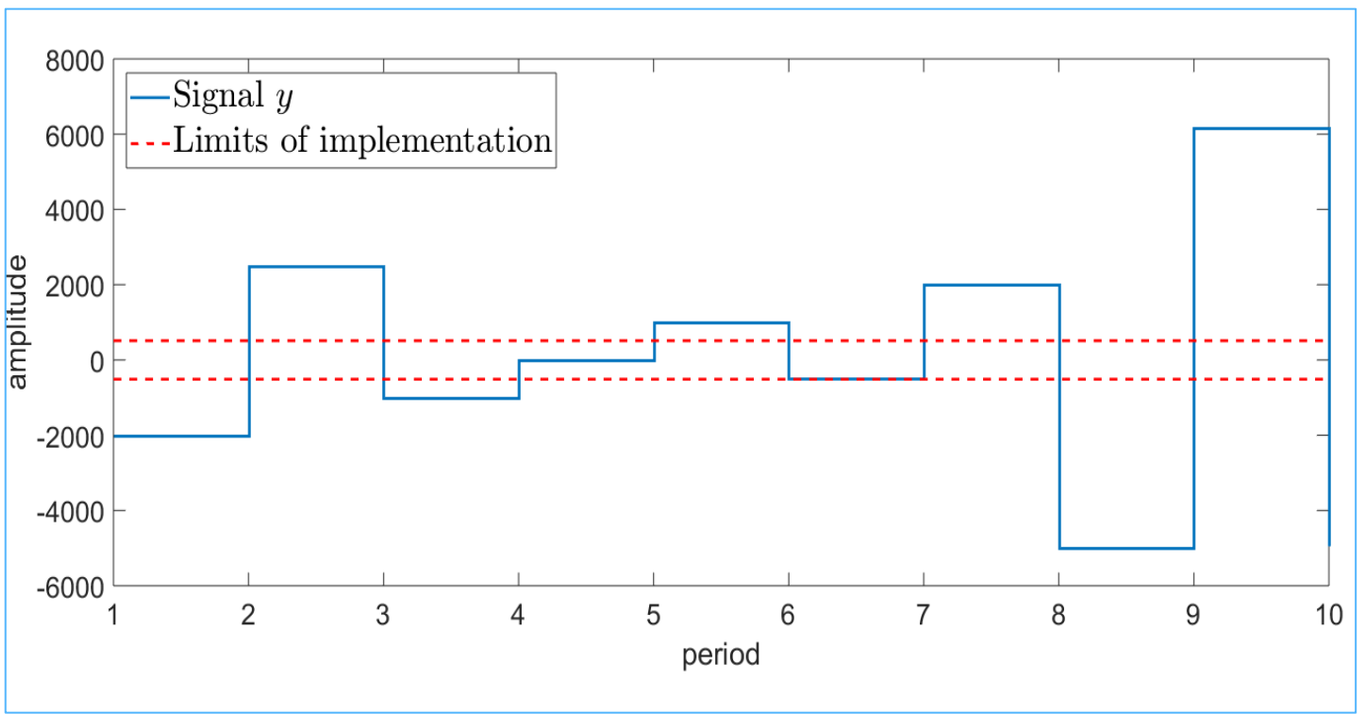
# DSValidator Features

- **Validation Functions**
  - reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)
- **Properties**
  - checks and validates overflow, limit-cycle, stability and minimum-phase
- **Realization**
  - reproduces realization forms to validate overflow and limit-cycle (for direct and delta forms)
- **Numerical Functions**
  - performs the quantization process, select rounding and overflow mode, fixed-point operations and delta operator

# Graphical Functions

```
plot_limit_cycle(system)
```



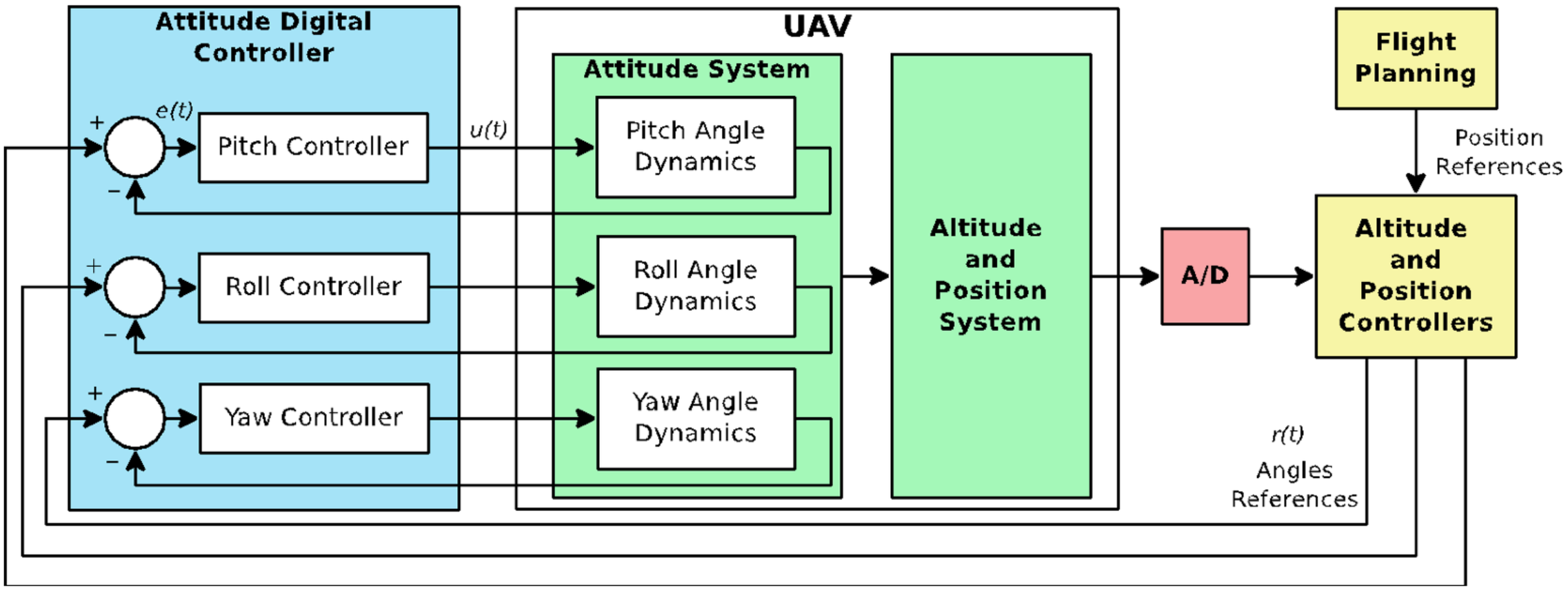

```
plot_overflow(system)
```

# DSValidator Usage

- **MATLAB Command Line:**
  - *validation*(path, property, ovmode, rmode, filename)
    - **path**
      - is the directory with the counterexample
    - **property**
      - **“m”** for minimum phase
      - **“s”** for stability
      - **“o”** for overflow
      - **“lc”** for limit cycle
    - **ovmode**
      - overflow mode: **wrap** or **saturate**
    - **rmode**
      - rounding mode: **round, float or ceil**
    - **filename**
      - represents the **.MAT filename**, which is **generated** after the **validation process**; by default, the .MAT file is named **digital_system**

# Case Study: Digital Controllers for UAV

- 11 digital controllers extracted from a **quadrotor unmanned aerial vehicle**
- **Overflow, minimum-phase, stability** and **limit-cycle**
- 8-, 16- and 32-bit
- DFI, DFII and TDFII



# Experimental Evaluation

- **RQ1 (performance)** do the executable test cases take considerably **less effort than verification**?
- **RQ2 (sanity check)** are the counterexamples sound and can their **reproducibility** be confirmed?

| Property | CE Reproducible | CE Irreproducible | Time |
|---|---|---|---|
| **Overflow** | 24 | 0 | 0.190 s |
| **Limit Cycle** | 26 | 1 | 0.483 s |
| **Minimum-Phase** | 54 | 0 | 0.012 s |
| **Stability** | 54 | 0 | 0.188 s |

- For the **limit cycle** property:
  - it did not take into account overflow in intermediate operations to compute the system's output using the DFII realization form

# Github commit to fix the bug

**enabling overflow mode saturation for intermediate operations.**

Browse files

master v2.0.3

**lennonchaves** committed on Nov 4, 2016 — 1 parent 52bcfea — commit 88e857bdbc74a7ce3c74d327e2a1e7a246fa48cc

Showing **2 changed files** with **9 additions** and **2 deletions**. Unified Split

6 ■■■■■ bmc/core/fixed-point.h View

```
@@ -303,6 +303,7 @@ void fxp_to_double_array(double f[], fxp_t r[], int N) {
303 303  fxp_t fxp_abs(fxp_t a) {
304 304          fxp_t tmp;
305 305          tmp = ((a < 0) ? -(fxp_t)(a) : a);
    306 +        tmp = fxp_quantize(tmp);
306 307          return tmp;
307 308  }
308 309
@@ -315,6 +316,7 @@ fxp_t fxp_abs(fxp_t a) {
315 316  fxp_t fxp_add(fxp_t aadd, fxp_t badd) {
316 317          fxp_t tmpadd;
317 318          tmpadd = ((fxp_t)(aadd) + (fxp_t)(badd));
    319 +        tmpadd = fxp_quantize(tmpadd);
318 320          return tmpadd;
319 321  }
```

Sign in now

# Conclusions and Future Work

- DSValidator **reproduces counterexamples** generated for digital controllers of a quadrotor attitude system
  - **implementation aspects**
  - **stability**, **minimum-phase**, **limit-cycle** and **overflow**

# Conclusions and Future Work

- DSValidator **reproduces counterexamples** generated for digital controllers of a quadrotor attitude system
  - **implementation aspects**
  - **stability**, **minimum-phase**, **limit-cycle** and **overflow**
- There is no other automated MATLAB toolbox that can reproduce counterexamples for digital system generated by verifiers
  - identify the reason why the counterexample cannot be reproduced

# Conclusions and Future Work

- DSValidator **reproduces counterexamples** generated for digital controllers of a quadrotor attitude system
  - **implementation aspects**
  - **stability**, **minimum-phase**, **limit-cycle** and **overflow**
- There is no other automated MATLAB toolbox that can reproduce counterexamples for digital system generated by verifiers
  - identify the reason why the counterexample cannot be reproduced
- As future work, we expect to contribute to digital system validation by supporting further verifiers (e.g., Polyspace)
  - Simulate the **hybrid dynamics** over the continuous time

# DSValidator available at: http://dsverifier.org/