

# **Independent Testing**

**Hideki Nomoto  
JAXA/JAMSS**

# Purpose

- **To perform full-case verification (*"Robustness Test"*) of Flight SW FDIR logic**

# How?

- **By generating test cases necessary for formal proof of the SW**
- **By comparing between the results of flight SW test and our independent test for the generated cases**

# Why?

- **It is extremely expensive to perform "robustness test" by conventional manual testing**

# Strategy

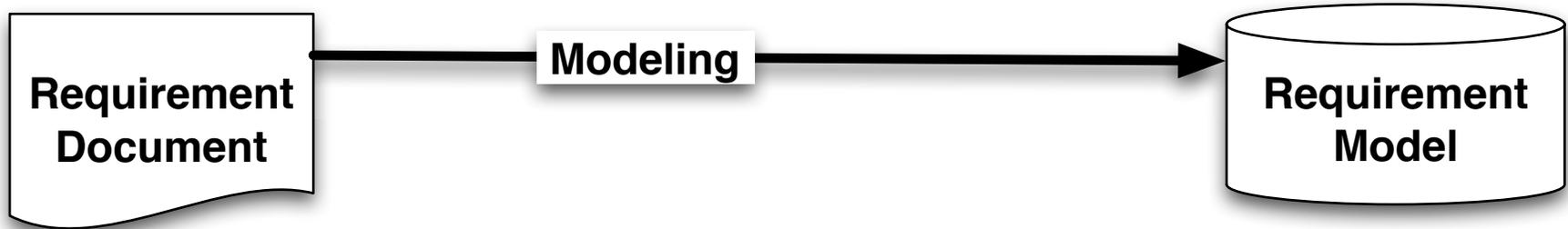
- 🌐 **Automatically generate "robustness test" cases from the requirement model built by IV&V**
- 🌐 **Conduct Monte Carlo Simulation of the flight SW for the test cases**
- 🌐 **Conduct same Monte Carlo Simulation of the auto-generated code from the design model built by IV&V**
- 🌐 **Perform auto-comparison of the two results using the criteria generated from requirement model built by IV&V**

# Big Picture of the Full Case Verification

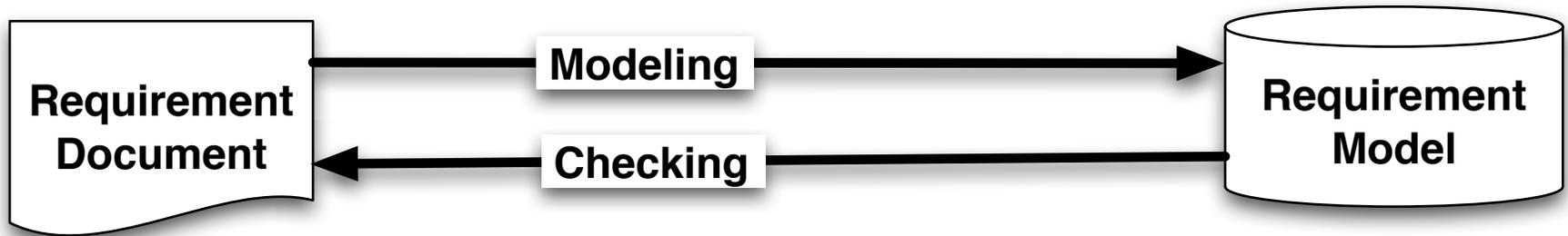
A rectangular box with a black border and a white background, containing the text "Requirement Document". The bottom edge of the box is slightly curved.

Requirement  
Document

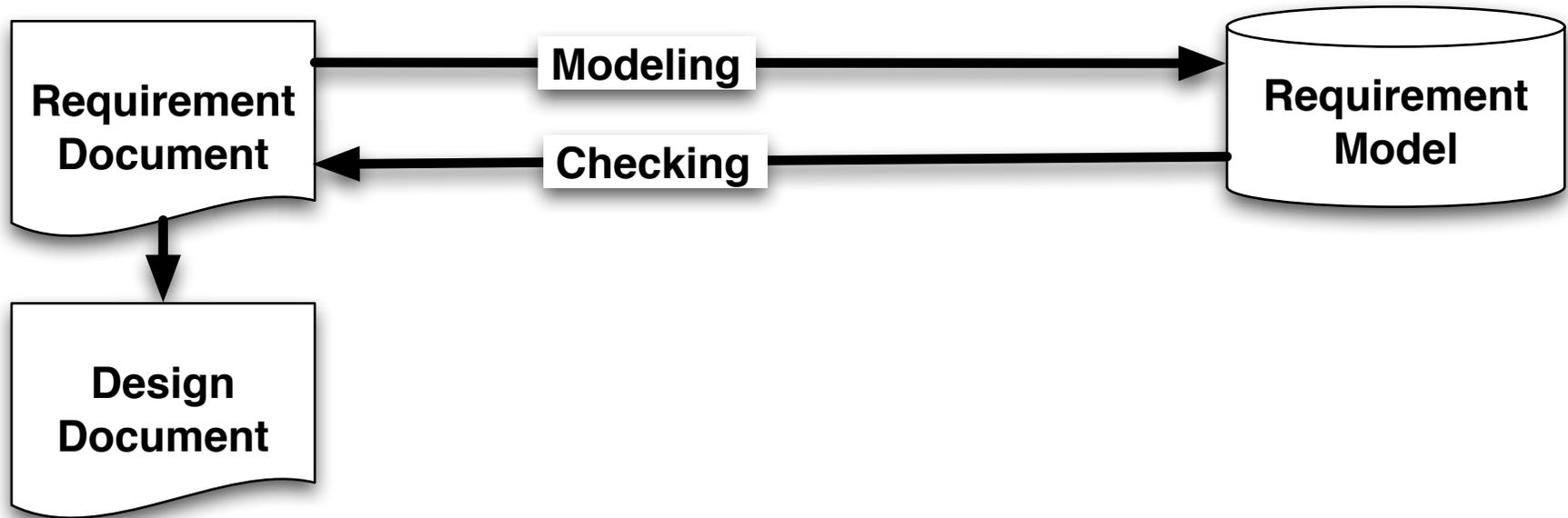
# Big Picture of the Full Case Verification



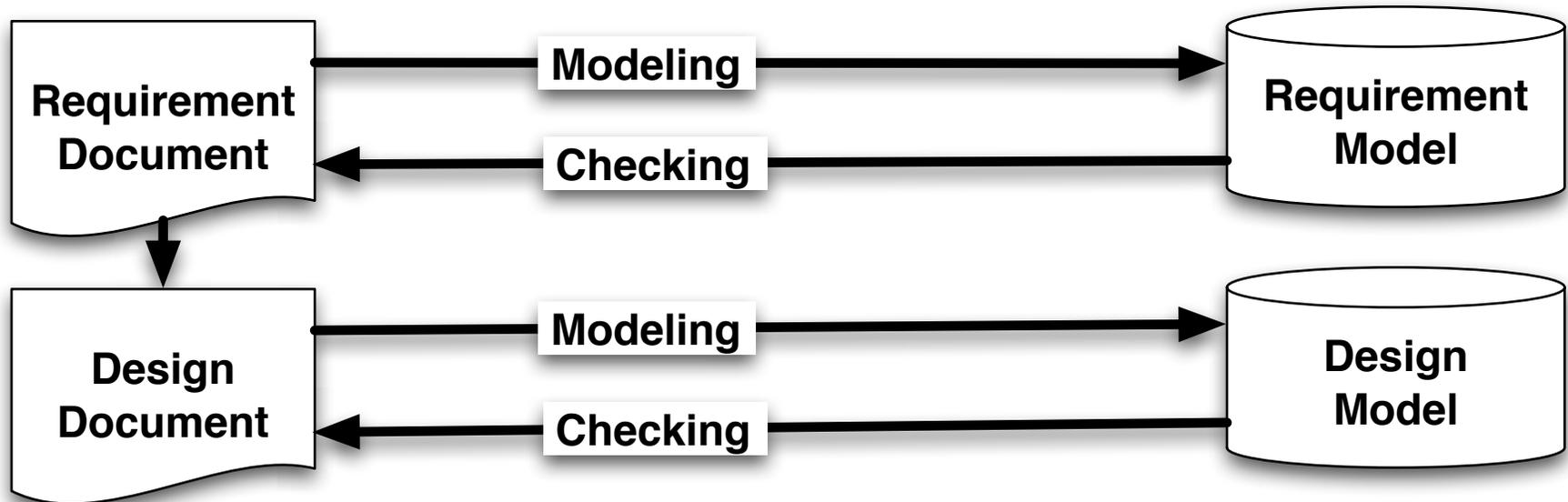
# Big Picture of the Full Case Verification



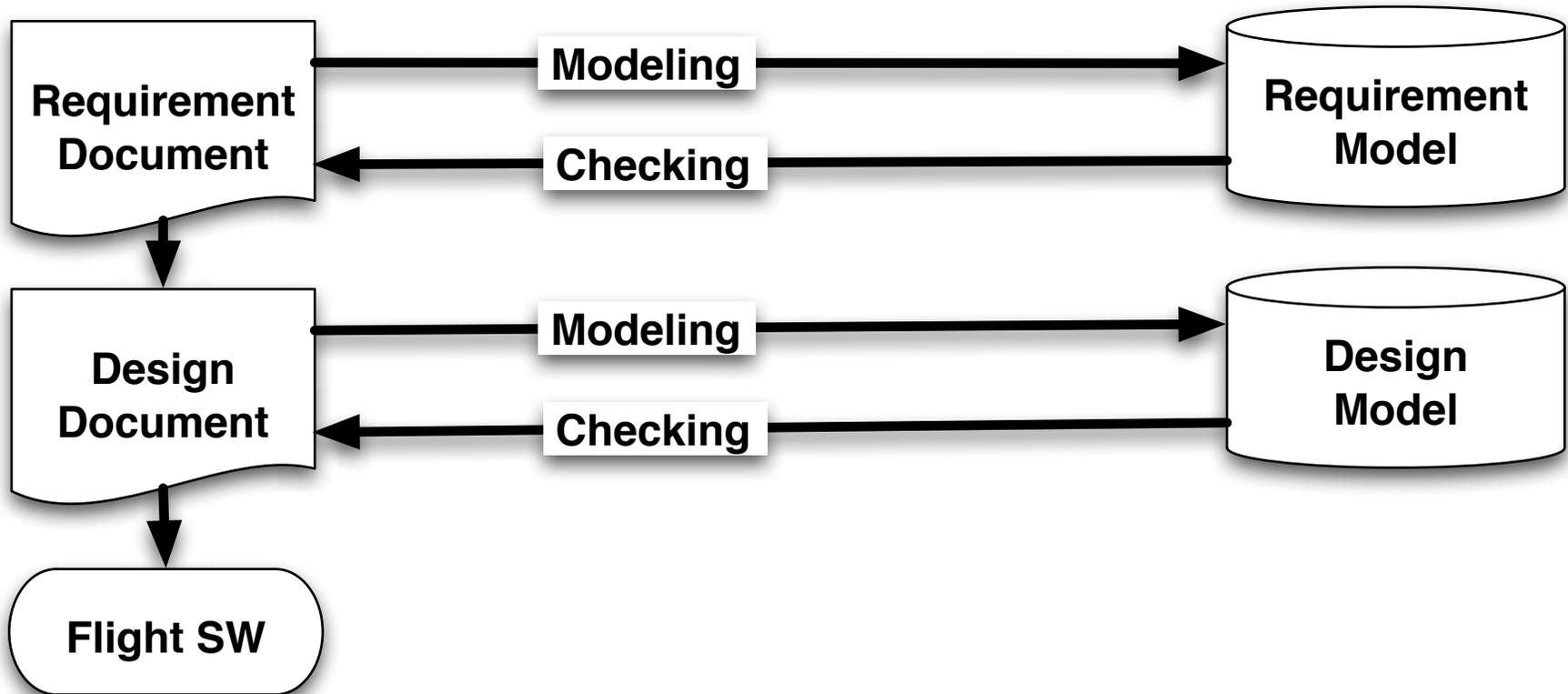
# Big Picture of the Full Case Verification



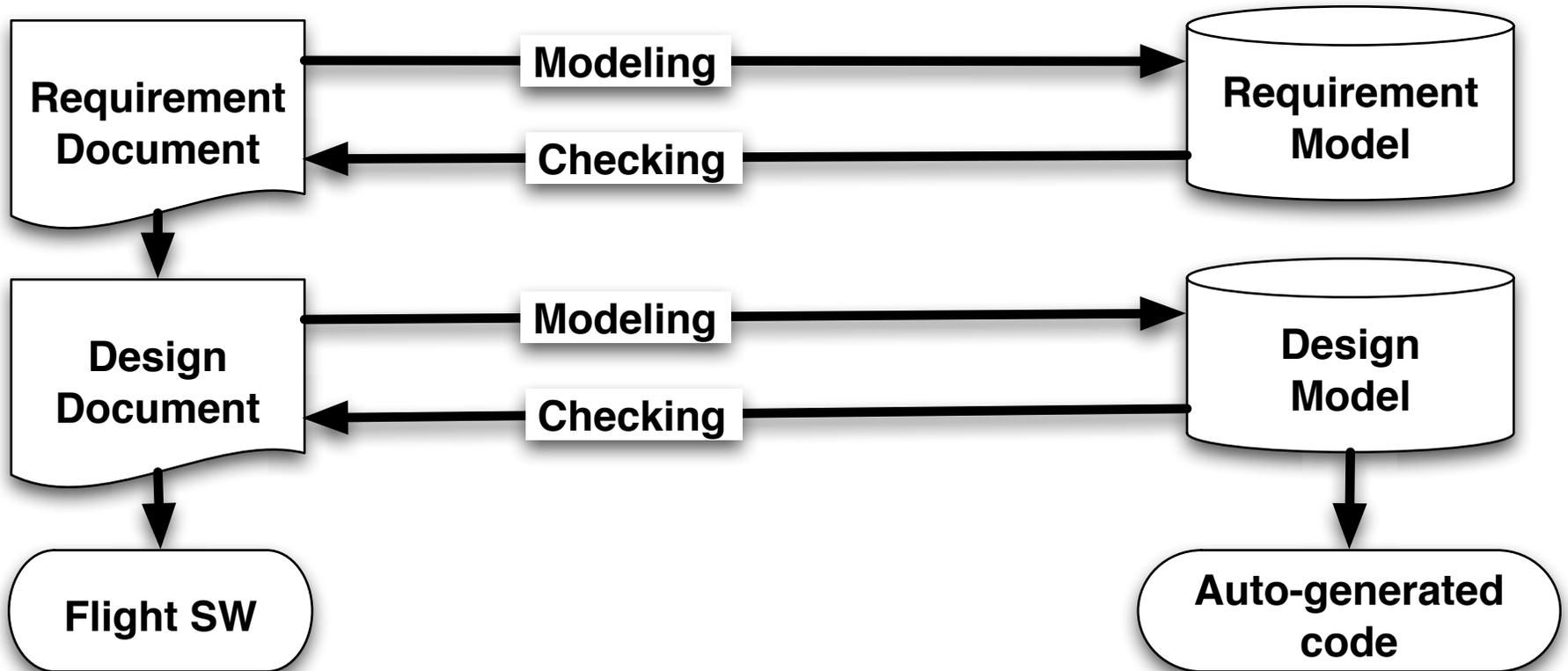
# Big Picture of the Full Case Verification



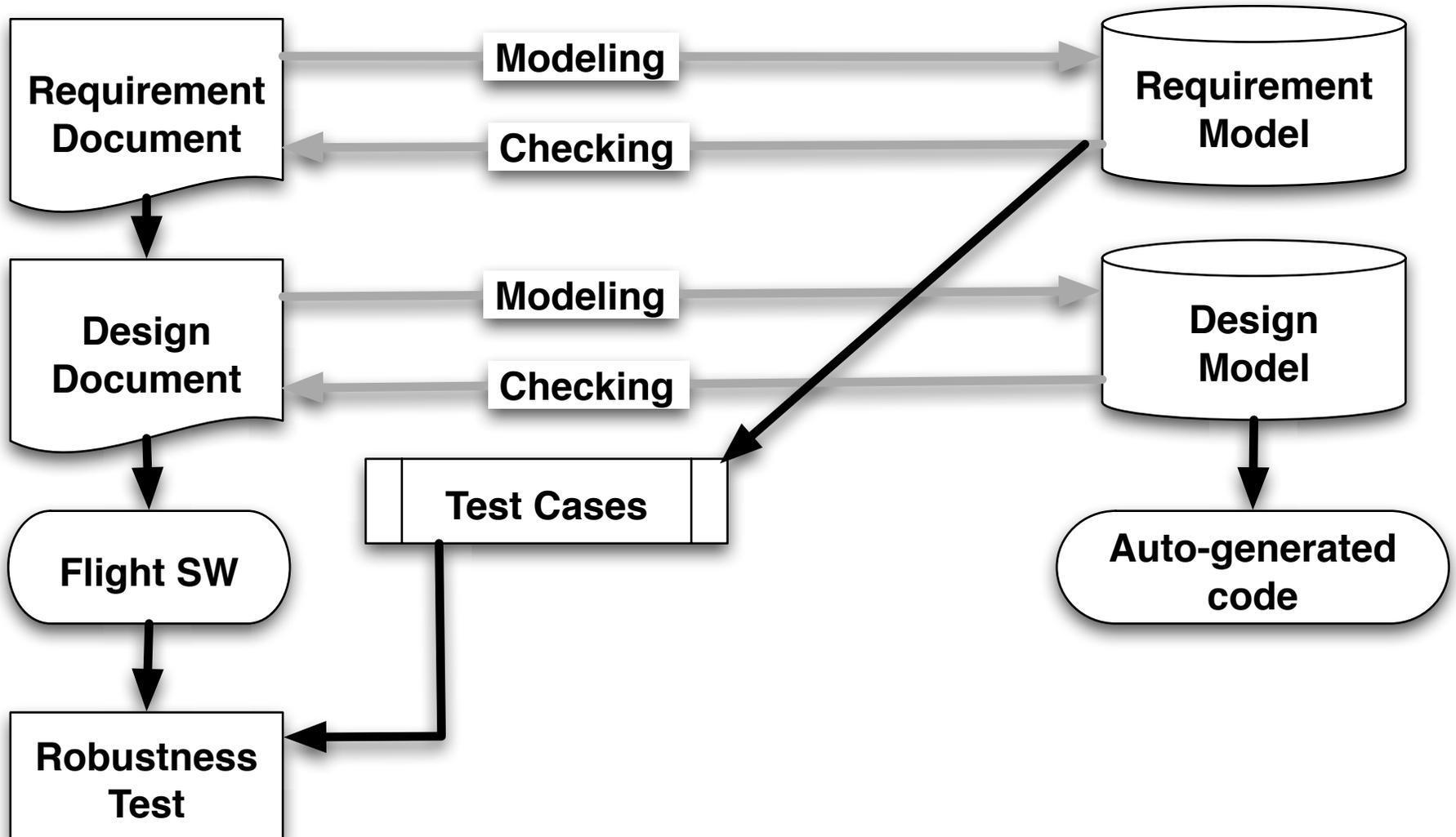
# Big Picture of the Full Case Verification



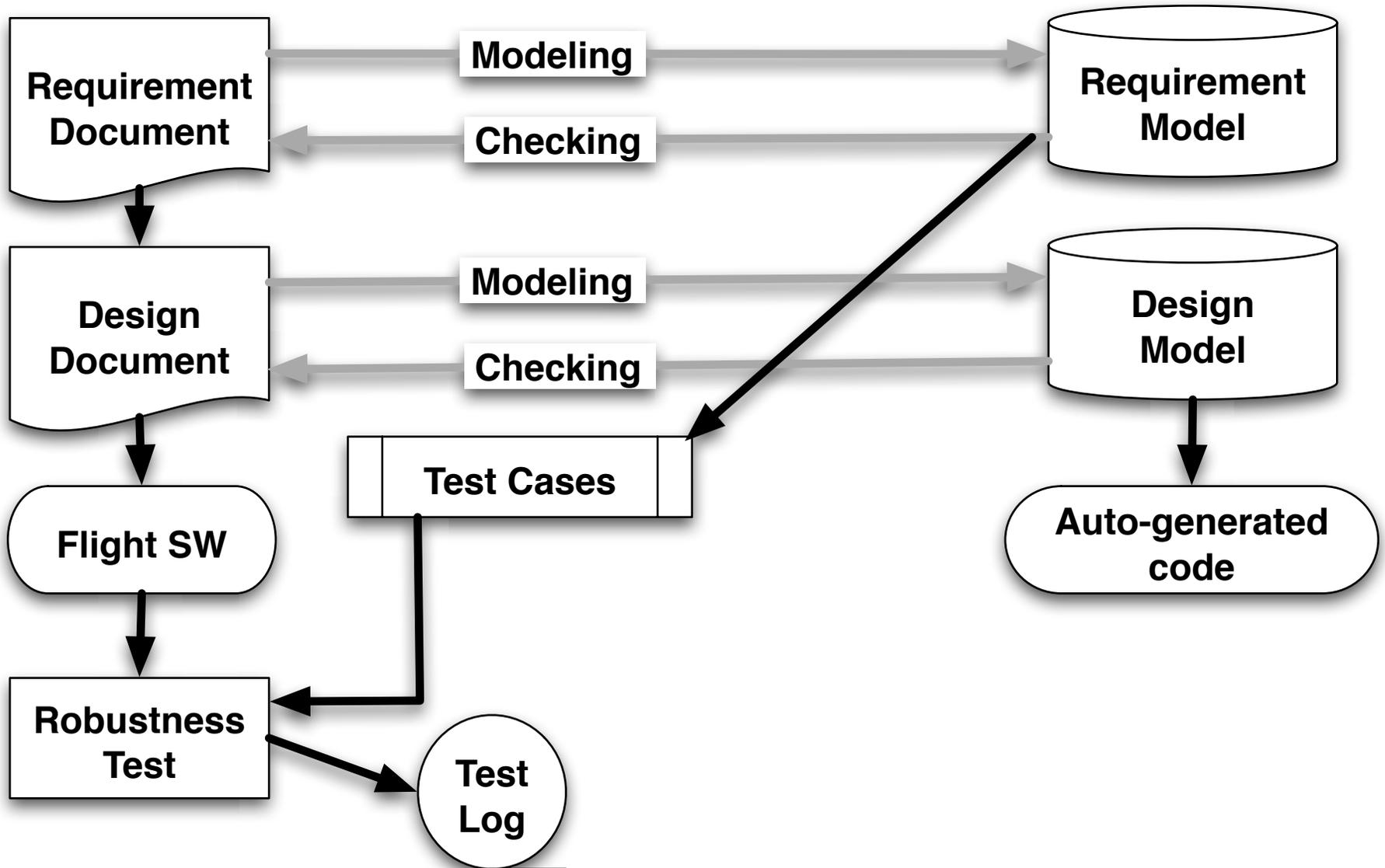
# Big Picture of the Full Case Verification



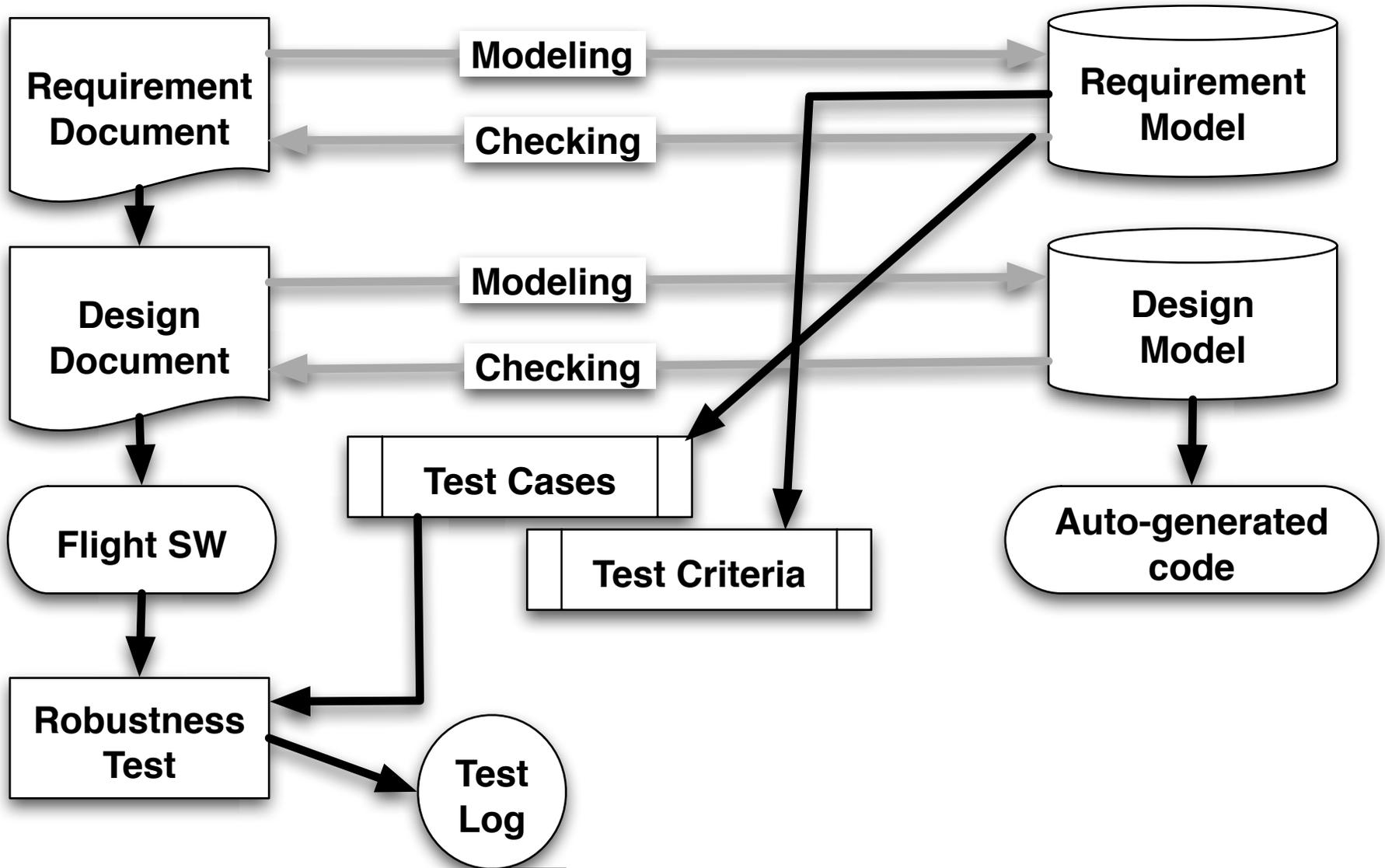
# Big Picture of the Full Case Verification



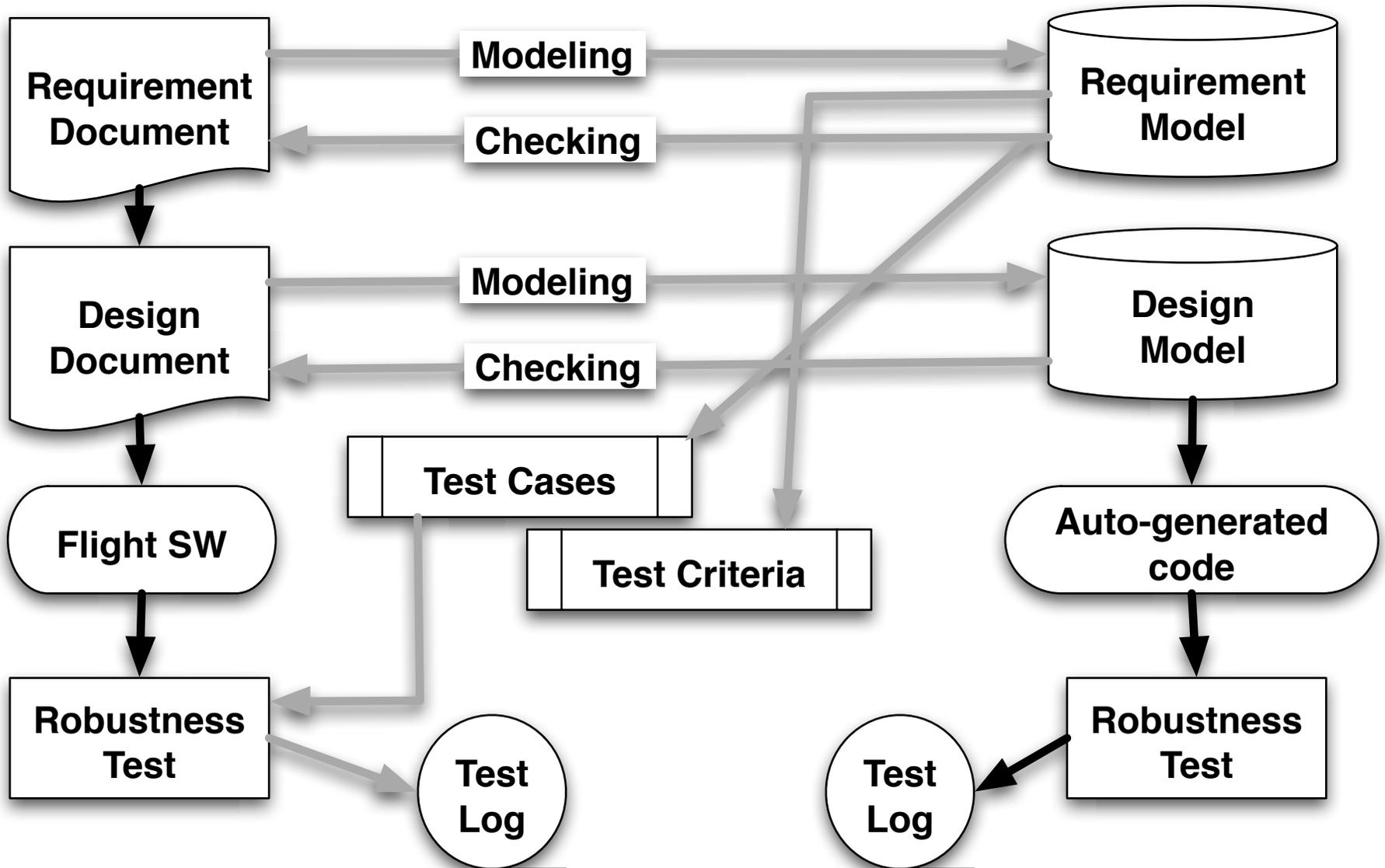
# Big Picture of the Full Case Verification



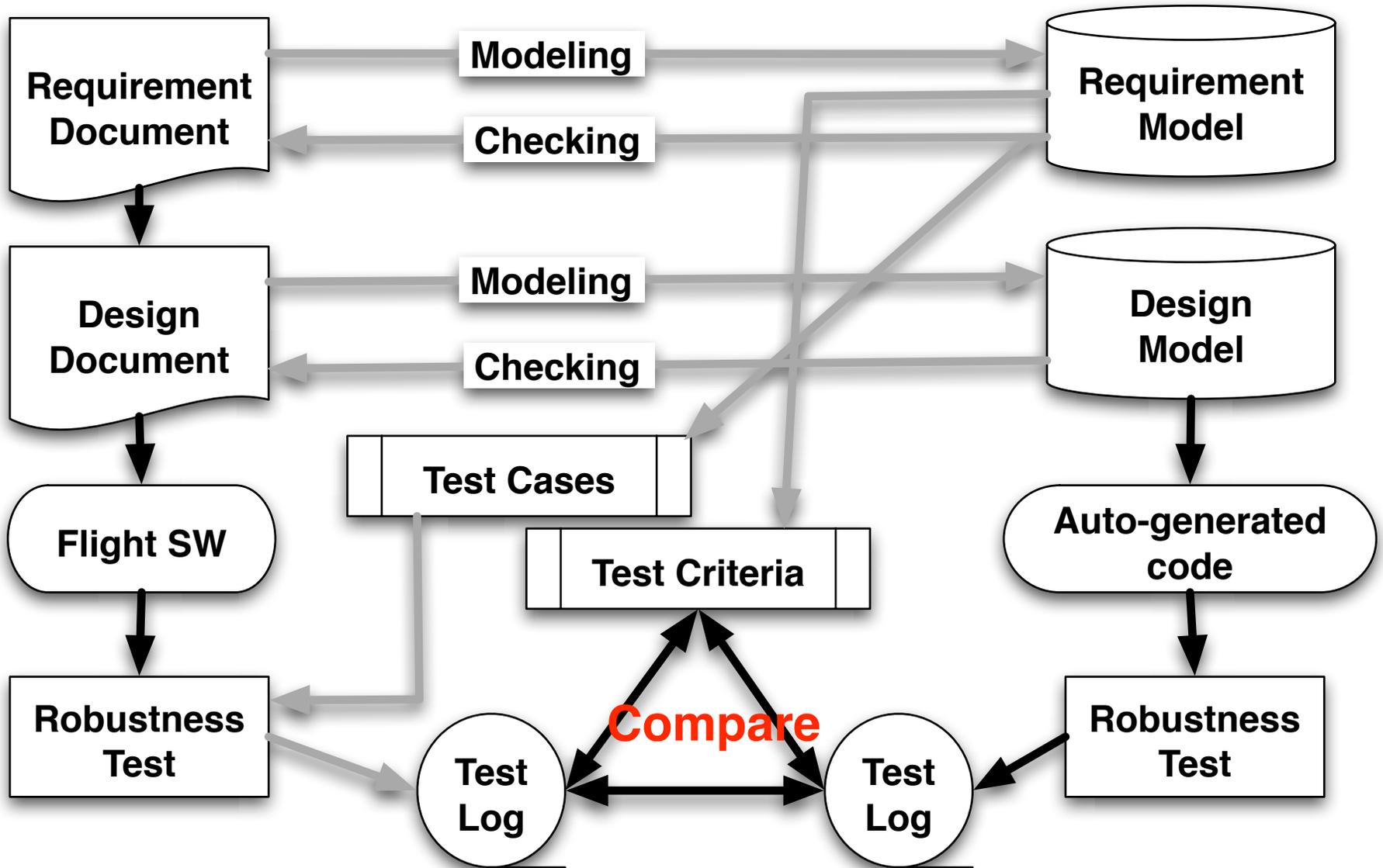
# Big Picture of the Full Case Verification



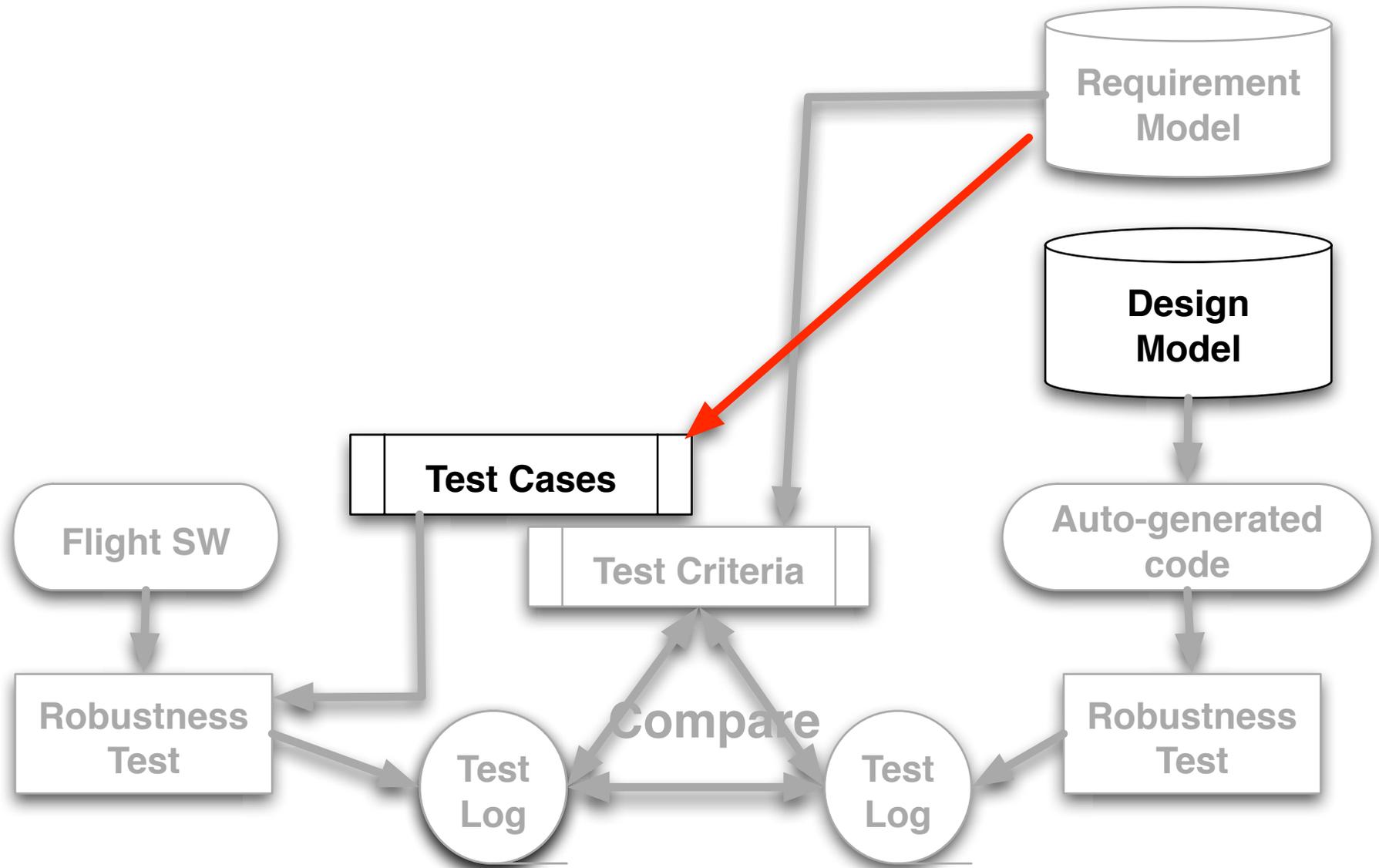
# Big Picture of the Full Case Verification



# Big Picture of the Full Case Verification



# Generating Test Cases



# Generating Test Cases

## Every Possible Combination of 2 Failures

Full path verification for 2FT system safety

# Generating Test Cases

- ✚ **Every Possible Combination of 2 Failures**

Full path verification for 2FT system safety

- ✚ **Every Possible Combination of Timing**

Monte Carlo randomizing of fault injection timing

# Generating Test Cases

- ✚ **Every Possible Combination of 2 Failures**

Full path verification for 2FT system safety

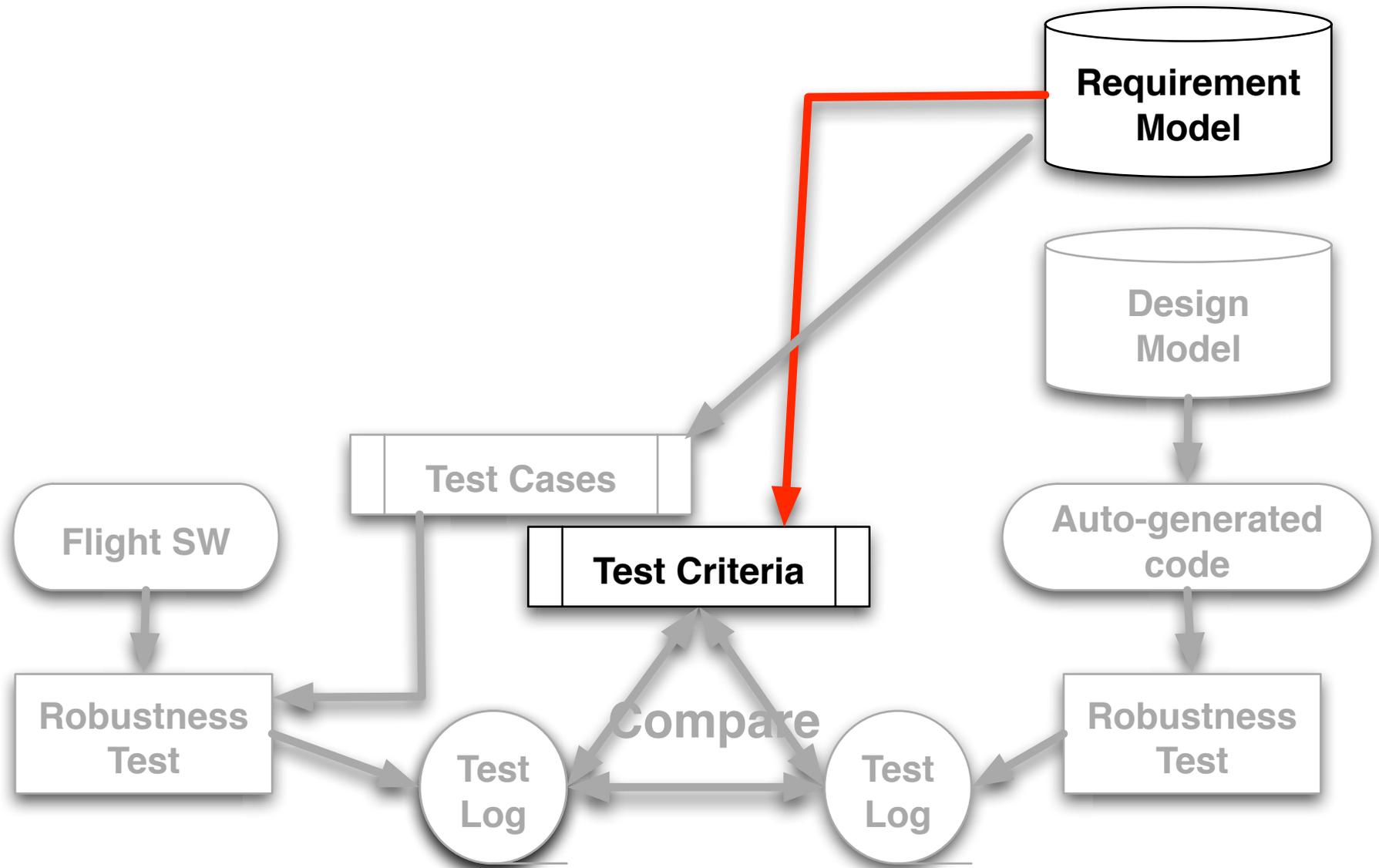
- ✚ **Every Possible Combination of Timing**

Monte Carlo randomizing of fault injection timing

- ✚ **Every Possible variation of Data Dispersion**

Monte Carlo randomizing of input data dispersion

# Generating Test Criteria



# Generating Test Criteria

- ✚ **Identification of state variables and their values to be checked**

Define sets of state values required to be observed with the given combinations of failures

- ✚ **Auto generation from requirement specification**

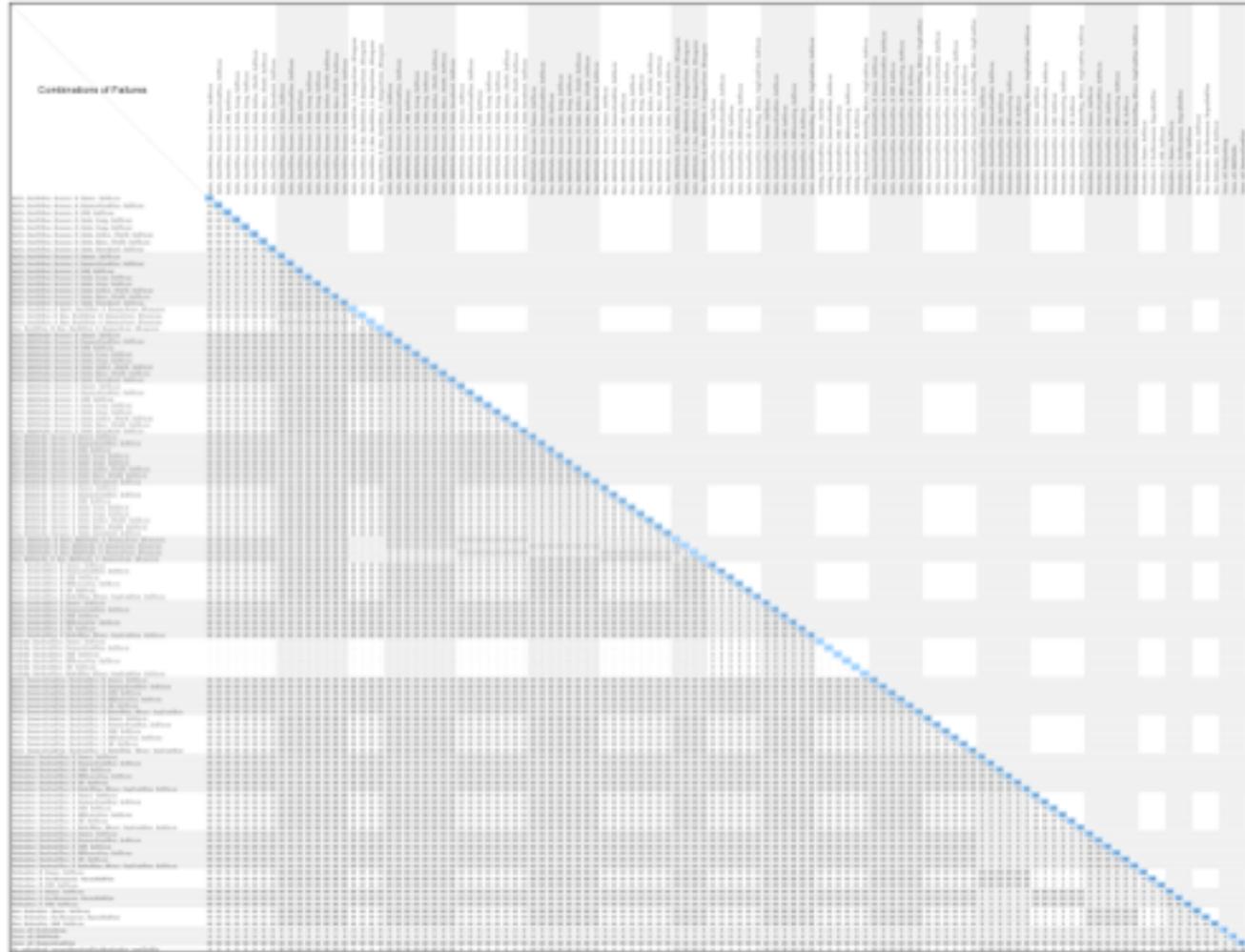
Generate state definition (see next chart) of the system with the given combinations of failures

# Generating Test Criteria

FDIR Requirement

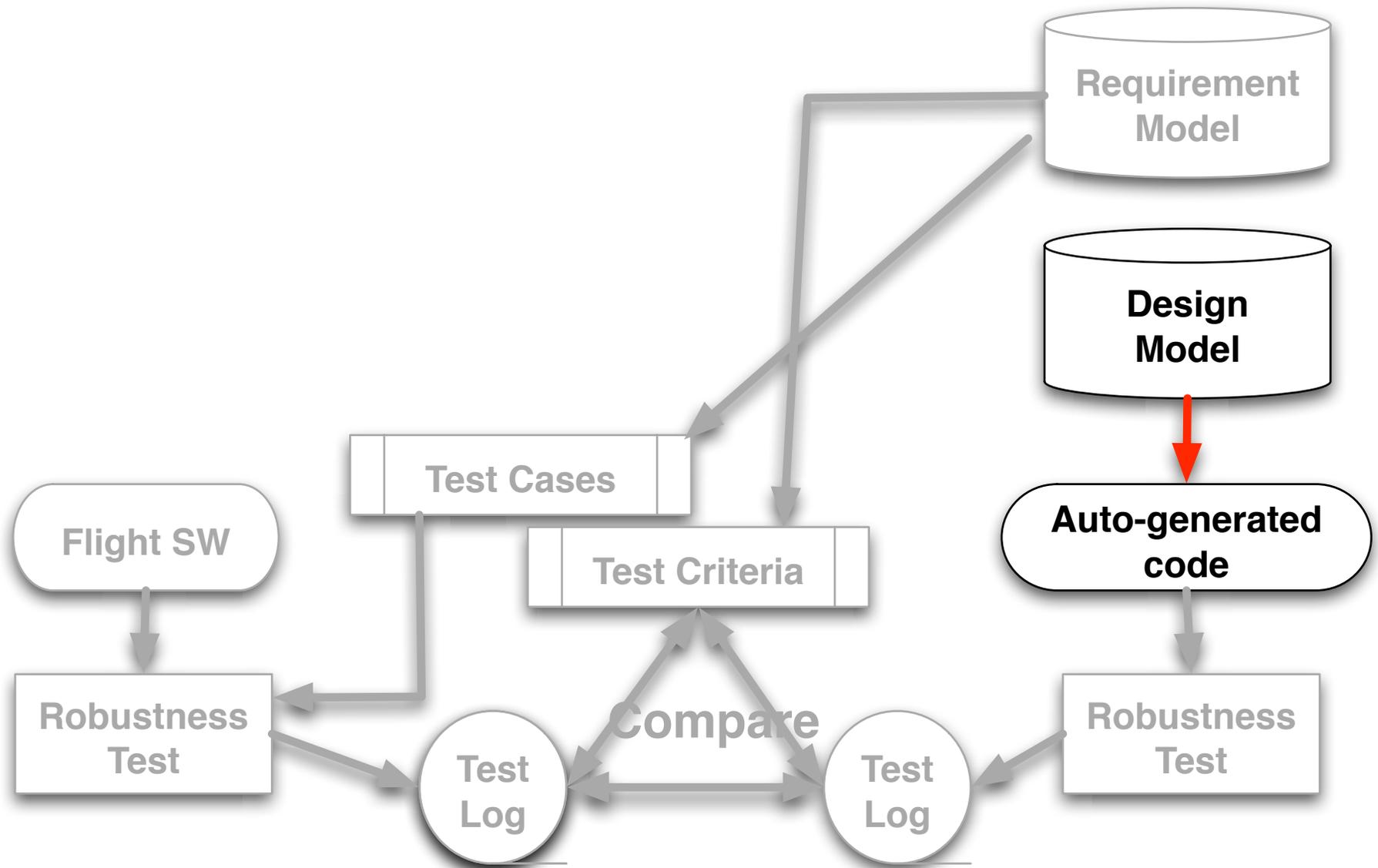
Failure Modes

Failure Modes





# Auto Code Generation from Design Model



# Auto Code Generation from Design Model

Attitude

State Value

DEFINITION

= Nominal

IMU is functional
GPS is ifunctional
Rate < 0.1deg/s

T  
T  
T

= Off-Nominal

IMU is functional
GPS is ifunctional
Rate < 0.1deg/s

F	*	*
*	F	*
*	*	F

```
boolean Attitude ()
```

```
{
```

```
  if (IMU())
```

```
    cnd[0] = "T";
```

```
  else cnd[0] = "F";
```

```
  if (GPS())
```

```
    cnd[1] = "T";
```

```
  else cnd[1] = "F";
```

```
  if (Rate()<0.1)
```

```
    cnd[2] = "T";
```

```
  else cnd[2] = "F";
```

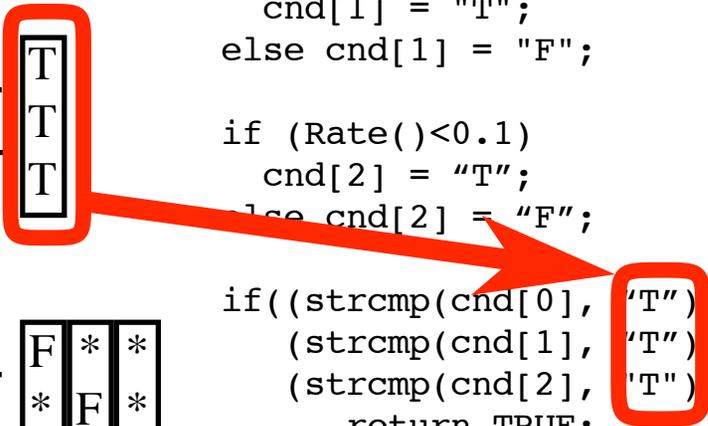
```
  if((strcmp(cnd[0], "T") &&
```

```
      (strcmp(cnd[1], "T") &&
```

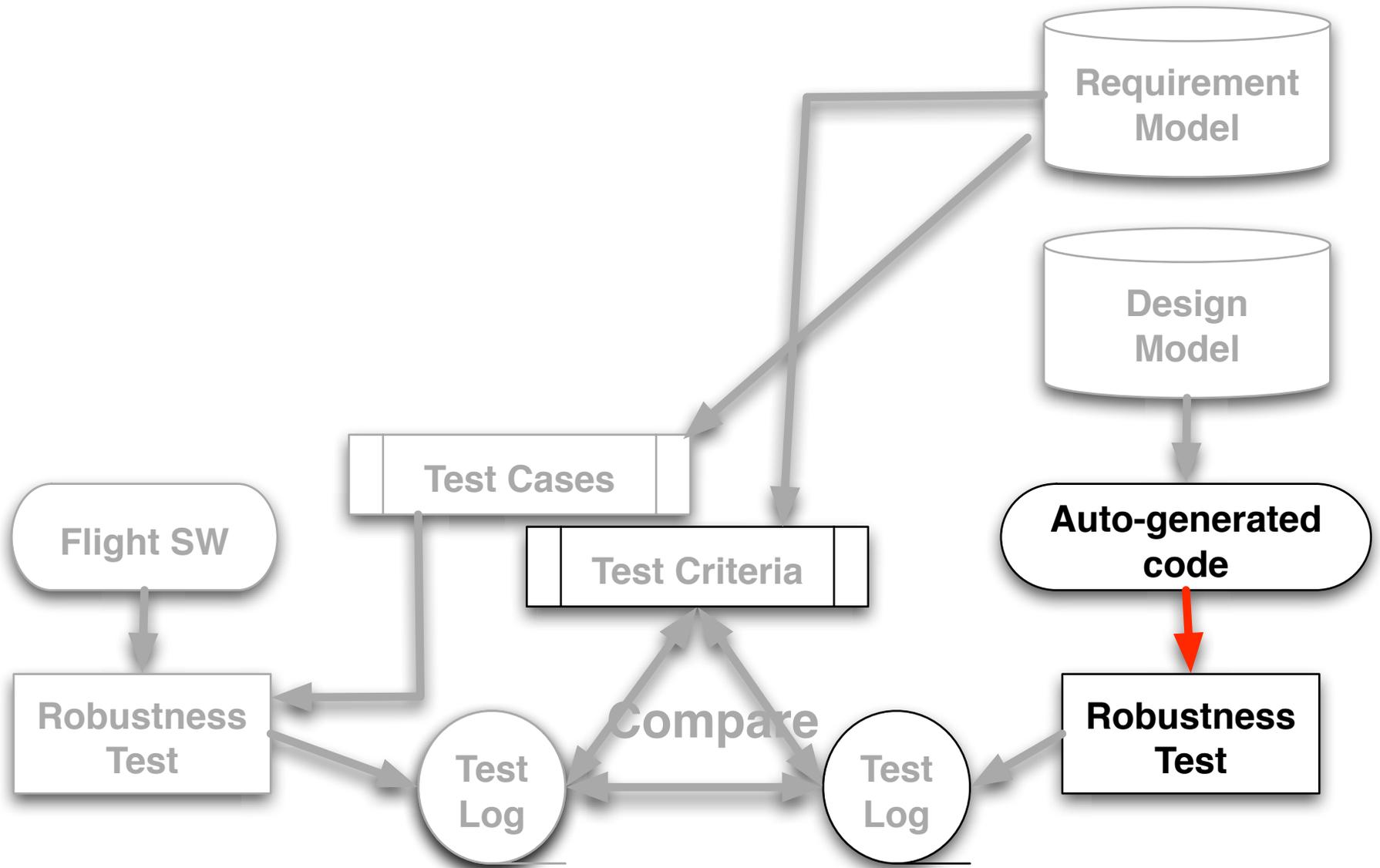
```
      (strcmp(cnd[2], "T"))
```

```
      return TRUE;
```

```
  else return FALSE;
```



# Auto Monte Carlo Simulation



# Auto Monte Carlo Simulation

The screenshot displays a simulation software interface. On the left is a code editor window titled 'Untitled' containing C code for a function named `set_Loss_of_Attitude`. The code includes variable declarations, static character tables for state transitions, and logic for handling children and status updates. Below the code, a message indicates the simulation result was written to `/tmp/Simulation_Result.sq`.

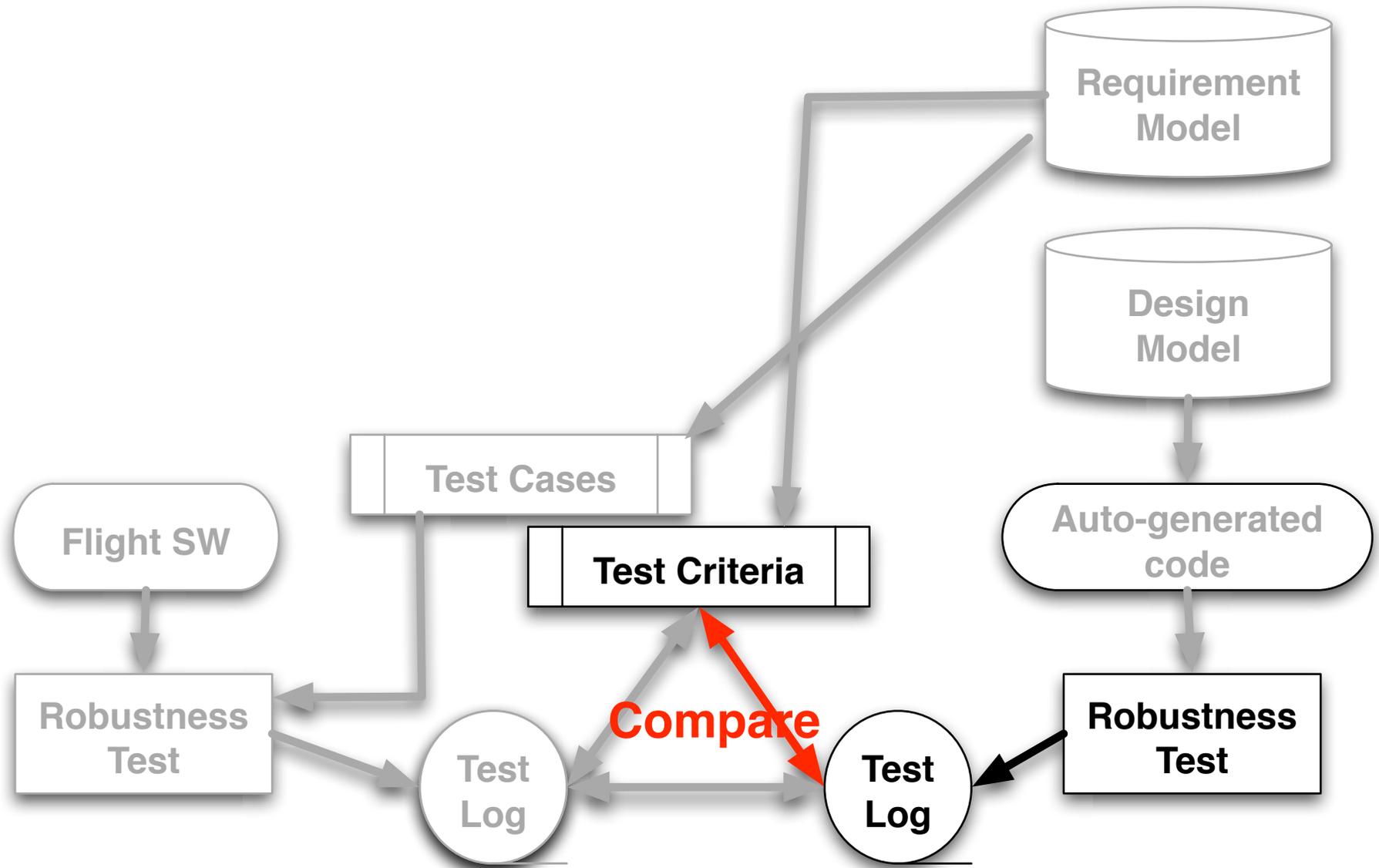
On the right is a window titled 'Simulation\_Result.sq' showing five stacked time-series plots. The x-axis for all plots represents time, with major ticks at 0, 5000, and 1e4. The y-axes represent the values of different variables:

- ESA\_Status**: A binary signal fluctuating between 0 and -1.
- Gyro\_Status**: A binary signal fluctuating between 0 and -1.
- ESA\_A**: A signal fluctuating between -5 and 5, showing a slight upward trend over time.
- ESA\_B**: A signal fluctuating between -5 and 5, showing a slight upward trend over time.
- Gyro\_A**: A signal fluctuating between -5 and 5.

At the bottom right, a text box displays the final values and previous values for the variables:

```
Prev Value of ESA_A : 0.000000  
ESA_B : 2.752218  
Last Value of ESA_B : 2.290000  
Prev Value of ESA_B : 2.280000  
Gyro_A : -3.894785  
Last Value of Gyro_A : 1.502517  
Prev Value of Gyro_A : 0.000000  
Gyro_B : 0.720477  
Last Value of Gyro_B : -0.287011  
Prev Value of Gyro_B : 0.000000  
Loss_of_Attitude : 1  
Last Value of Loss_of_Attitude : 1  
Prev Value of Loss_of_Attitude : 0
```

# Verify the Generated Code wrt Requirement Specification



# Compare the Auto-sim result with Flight SW test result

