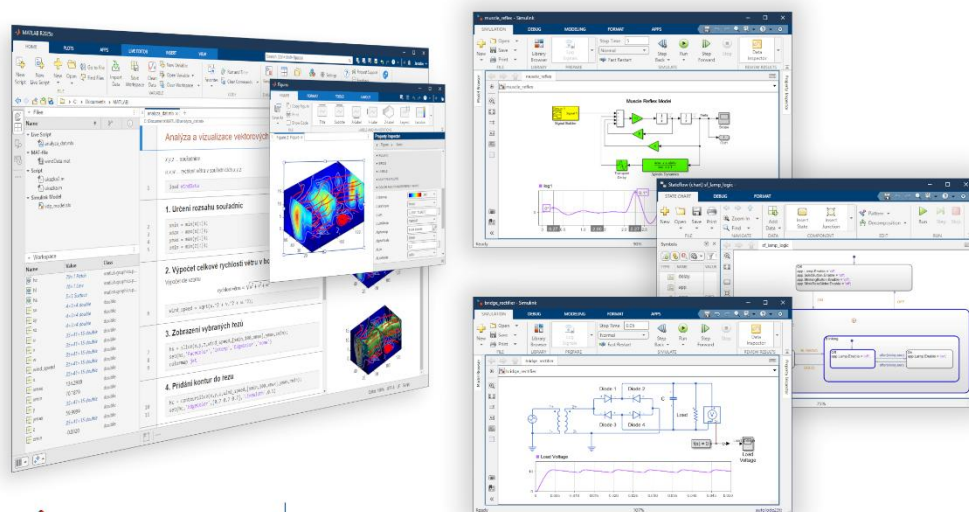


## MATLAB a Simulink Master Class: Vývoj řídicích systémů s využitím modelů

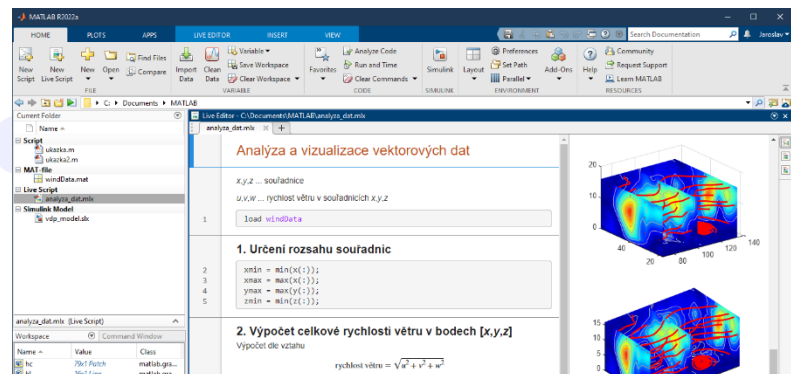


Jaroslav Jirkovský  
jirkovsky@humusoft.cz

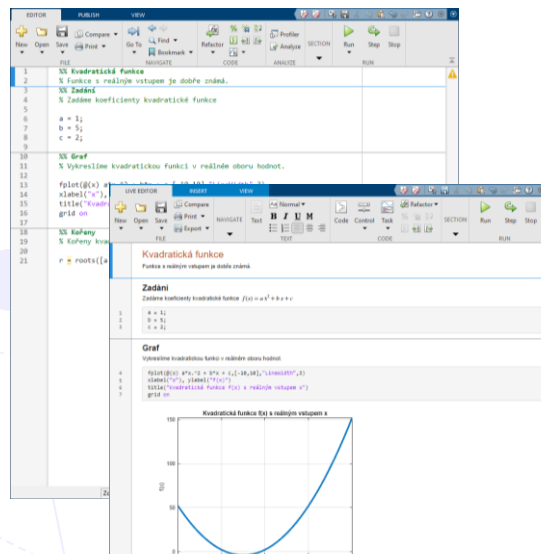
[www.humusoft.cz](http://www.humusoft.cz)  
[info@humusoft.cz](mailto:info@humusoft.cz)

[www.mathworks.com](http://www.mathworks.com)

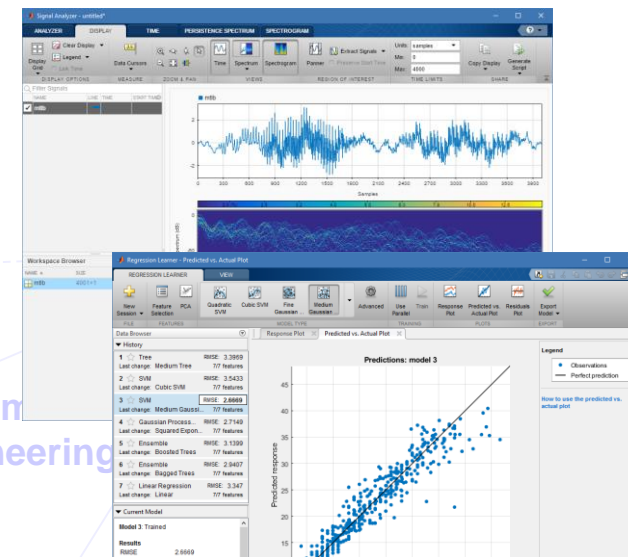
# Co je MATLAB



- inženýrský nástroj
- interaktivní prostředí
- vědecké a technické výpočty

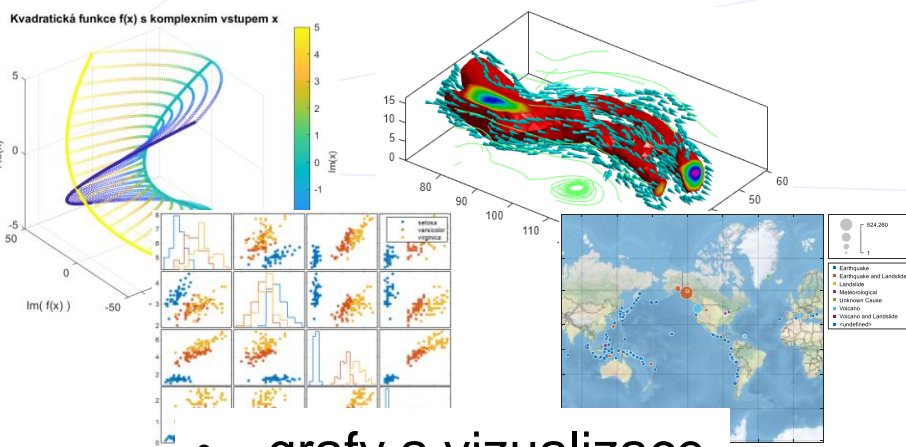


- výpočetní nástroje
- otevřený systém

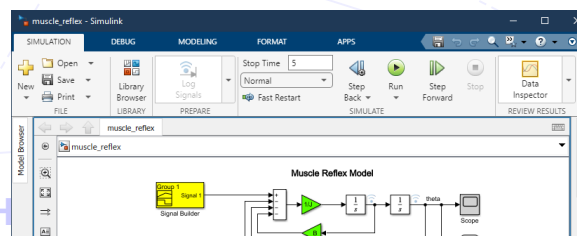


System Engineering

- grafické aplikace (Apps)



- grafy a vizualizace



- modelování systémů
- simulace a analýza
- Model-Based Design

Economics & Finance

- 100+ aplikačních knihoven
- 10 000+ výpočetních funkcí
- jednotná dokumentace

Industrial Systems

- propojení s externím hw / sw
- vývoj samostatných aplikací

# Kde se MATLAB používá ...



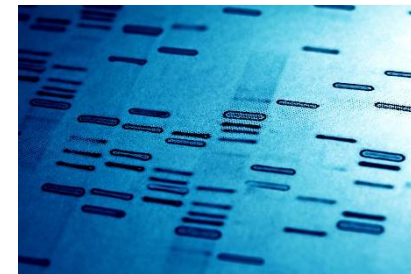
**Letecký průmysl**



**Automobilový průmysl**



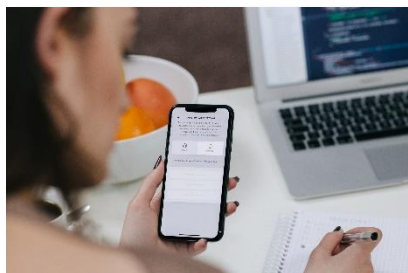
**Přírodní vědy**



**Biotechnologie a farmacie**



**Komunikace**



**Elektronika**



**Energetika**



**Finanční služby**



**Výrobní stroje**



**Medicína**



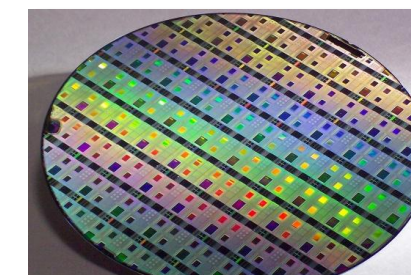
**Těžební průmysl**



**Neurovědy**



**Železnice a doprava**



**Polovodiče**



**Software a internet**

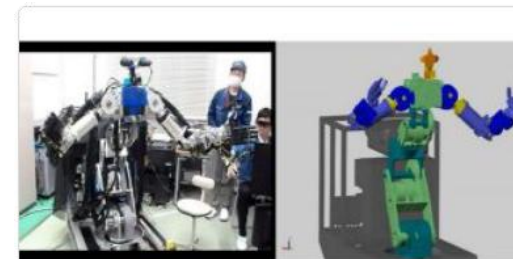
# Kde se MATLAB používá ... příklady v oblasti řídicích systémů



The Orion Spacecraft Is Headed to the Moon



Vanderhall Motor Works' Brawley: An All-Electric 303-Horsepower UTV



Advancing Robotics Design with Model-Based Design



Yanmar Slashes Diesel NOx Emissions with Deep Reinforcement Learning

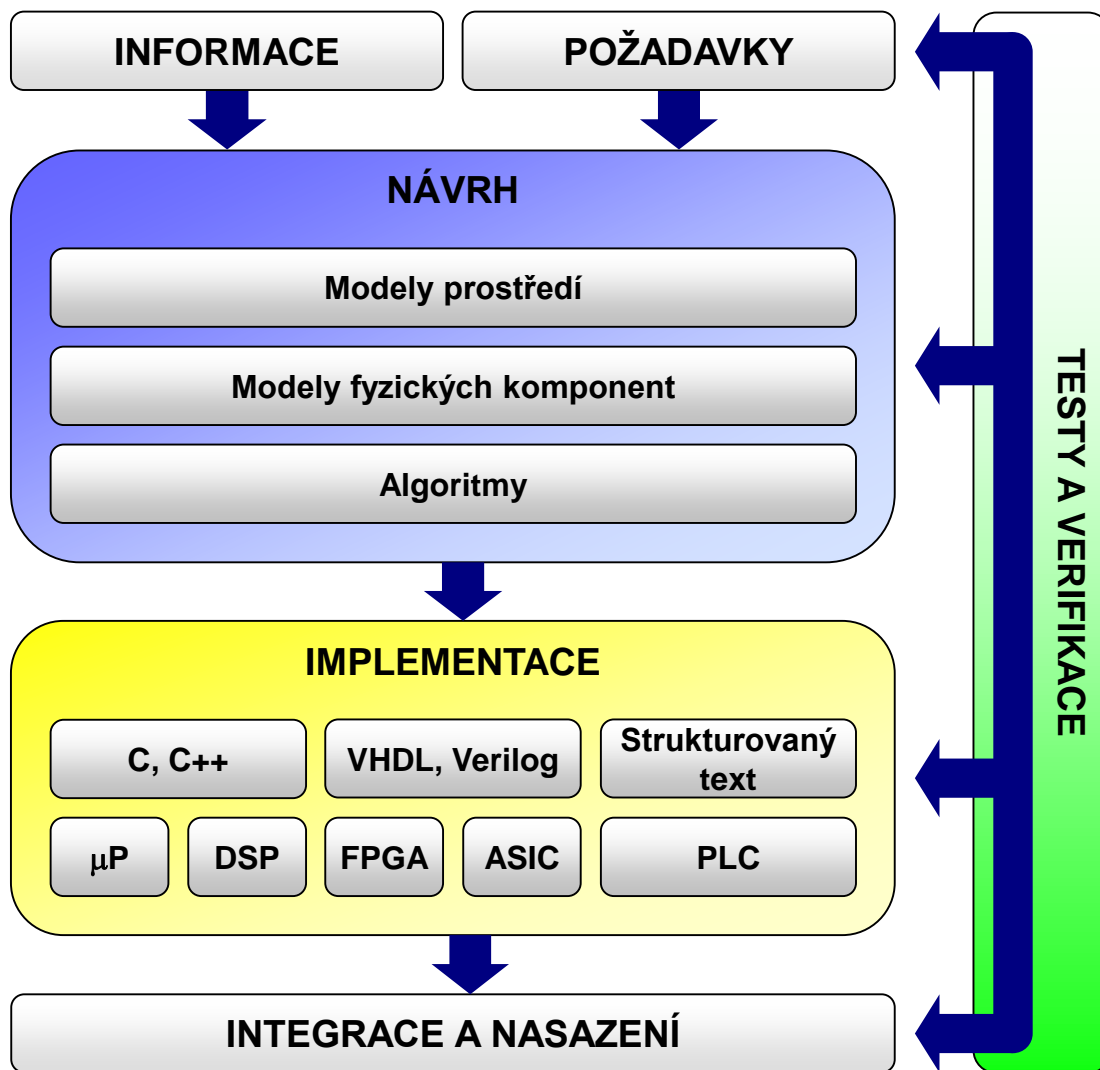


Dyson Uses Model-Based Design and Automatic Code Generation

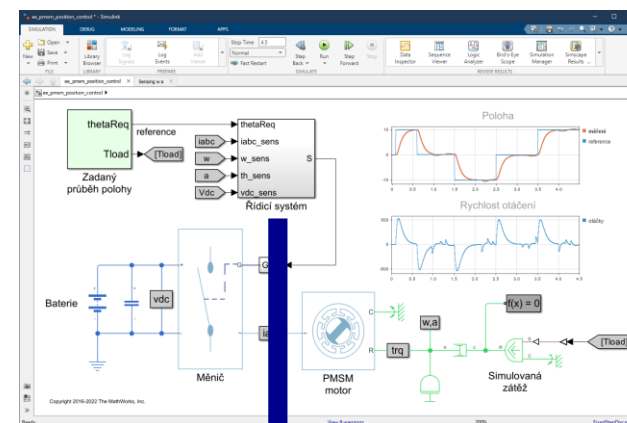


EVLO Energy Storage Accelerates Development of Energy Management...

# Vývoj metodou Model-Based Design



## Modelování, simulace a testování



Automatické generování kódu

```

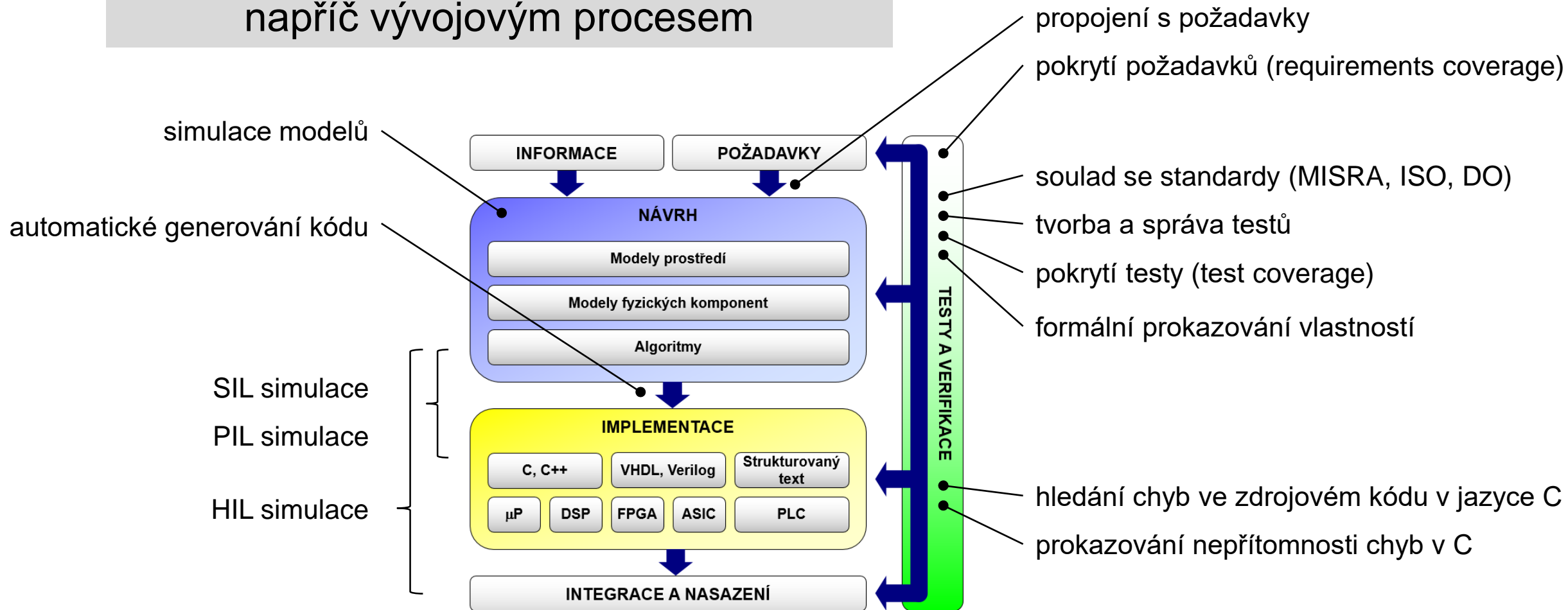
Code
Control_System.c
6 * Model version : 6.4
7 * Simulink Coder version : 9.9 (R2023a) 19-Nov-2022
8 * C/C++ source code generated on : Tue Apr 25 11:26:04 2023
9 *
10 * Target selection: ert-tlc
11 * Embedded hardware selection: Texas Instruments->C2000
12 * Code generation objectives: unspecified
13 * Validation result: not run
14 */
15
16 #include "Control_System.h"
17 #include "math.h"
18 #include "rt_modelinit.h"
19 #include "rtwtypes.h"
20 #include <string.h>
21
22 /* Block signals (default storage) */
23 #blockio_control_system Control_System_B;
24
25 /* Block states (default storage) */
26 #blockstates_control_system Control_System_S;
27
28 /* External inputs (root import signals with default storage) */
29 ExternalInputs_control_System Control_System_U;
30
31 /* External outputs (root outputs fed by signals with default storage) */
32 ExternalOutputs_control_System Control_System_Y;
33
34 /* Real-time model */
35 static #if_MCODE_control_System Control_System_M;
36 #if_MCODE_control_System *const control_System_M = &Control_System_M;
37
38 /* Model step function */
39 void control_System_step(void)
40 {
41     real32_T B1AS;
42     real32_T uB;
  
```

Nasazení

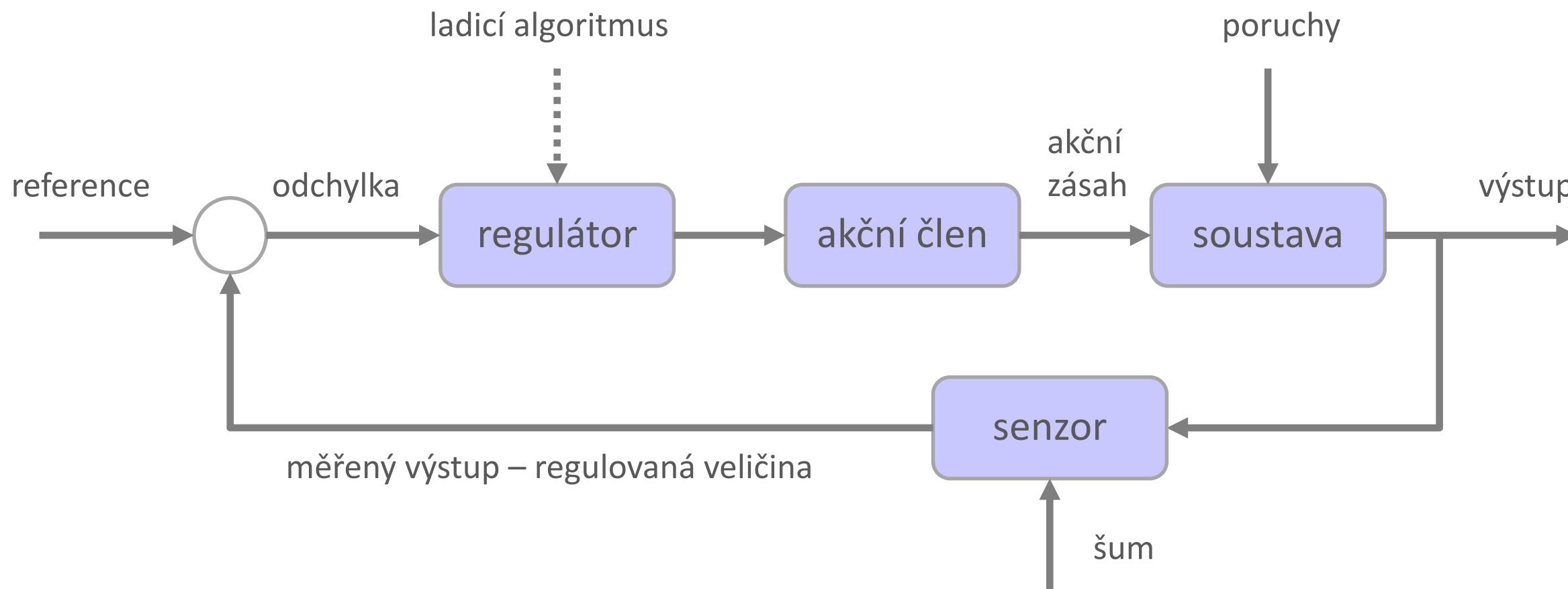


# Vývoj metodou Model-Based Design

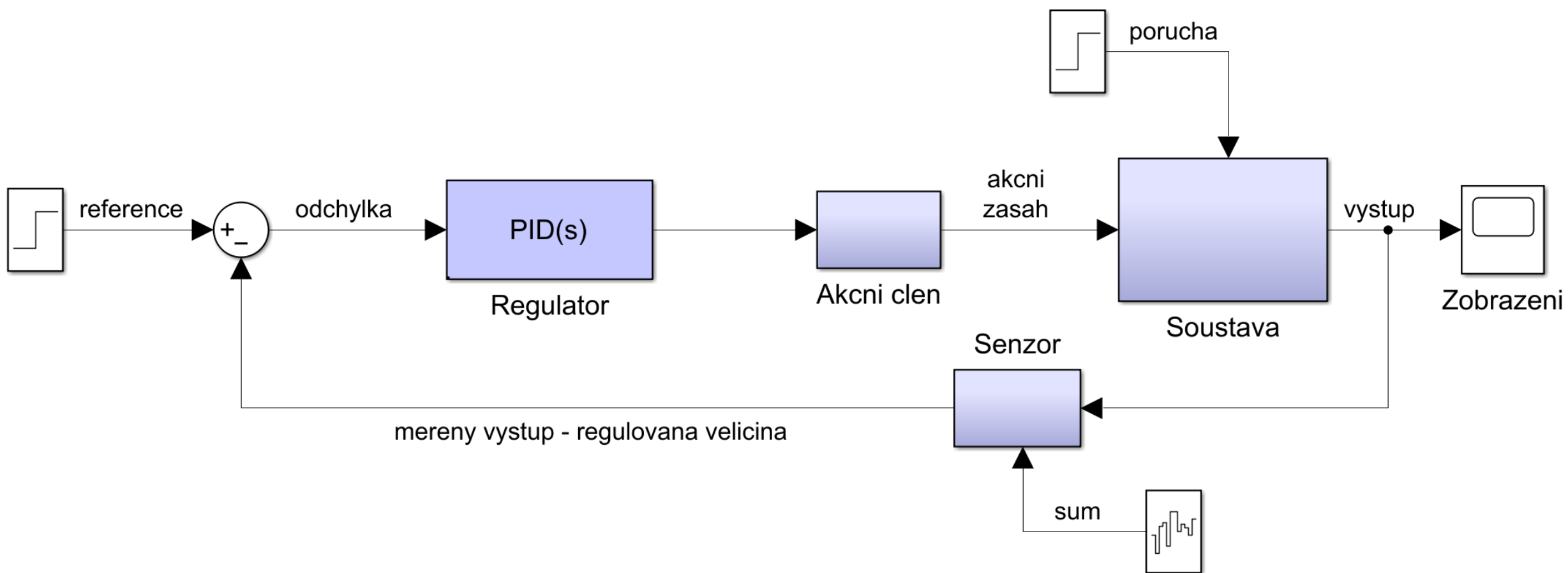
Systematické využití simulačních modelů  
napříč vývojovým procesem



# Schéma řídicího systému



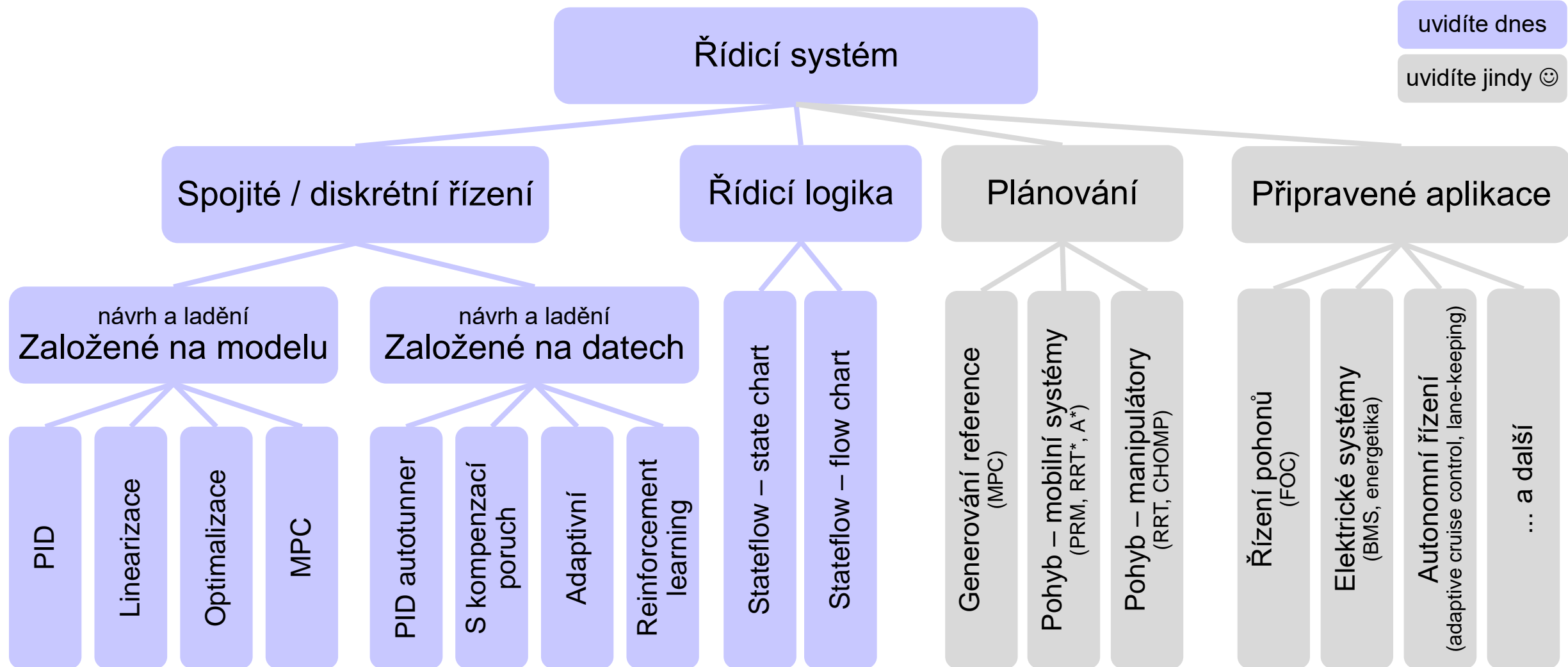
# Schéma řídicího systému – Simulink



# Možnosti návrhu řídicích systémů v prostředí MATLAB

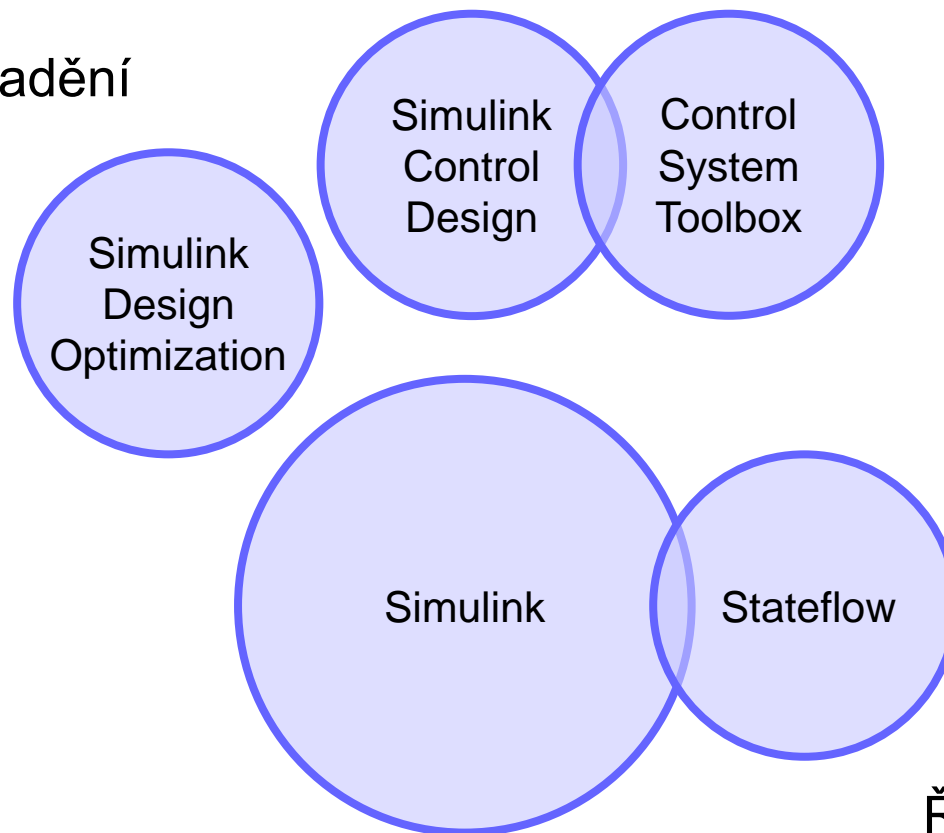
(výběr možností)

uvidíte dnes  
uvidíte jindy 😊



# MATLAB a Simulink pro návrh řídicích systémů

Optimalizace a ladění parametrů

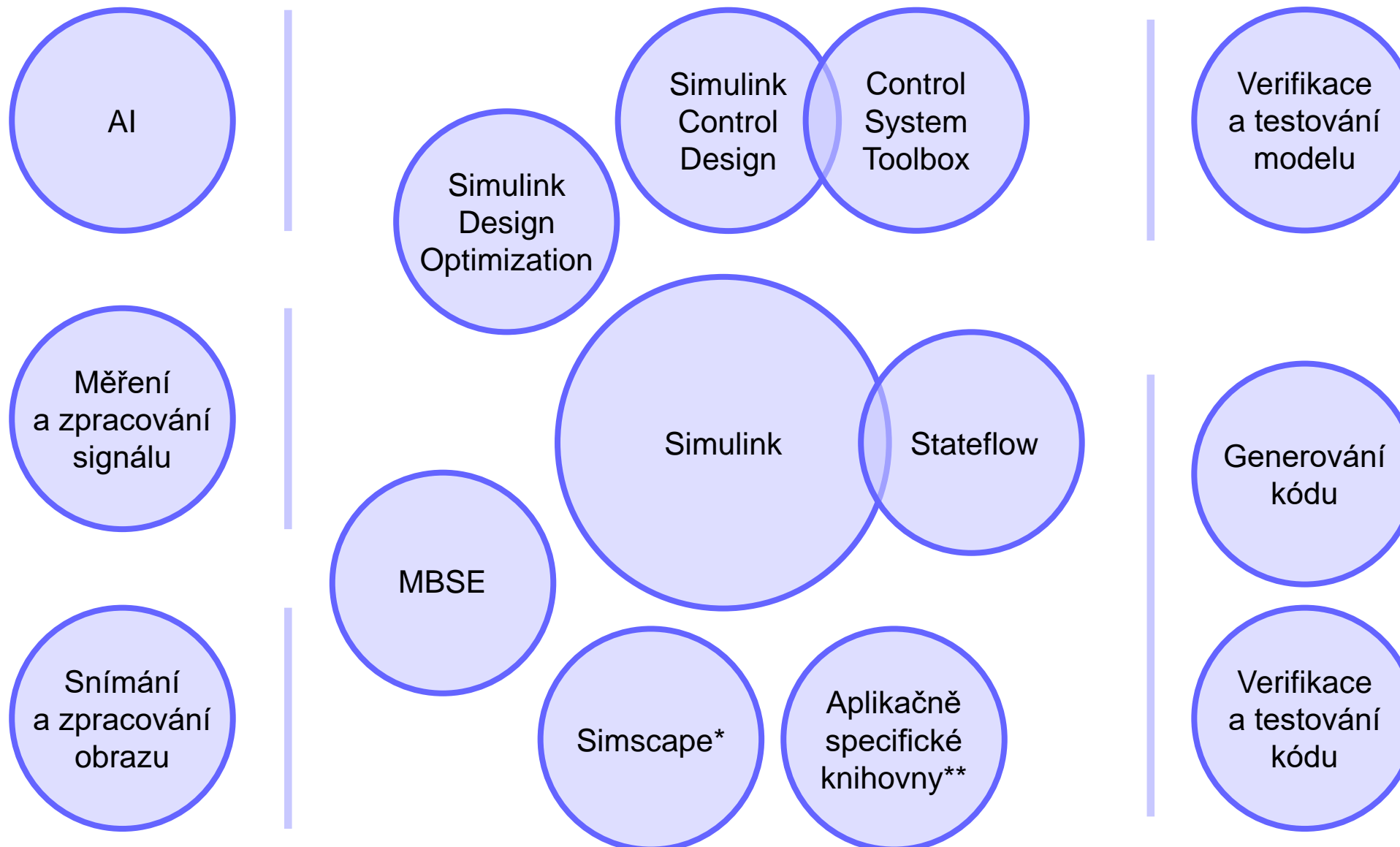


Návrh a ladění řídicích systémů, linearizace modelů

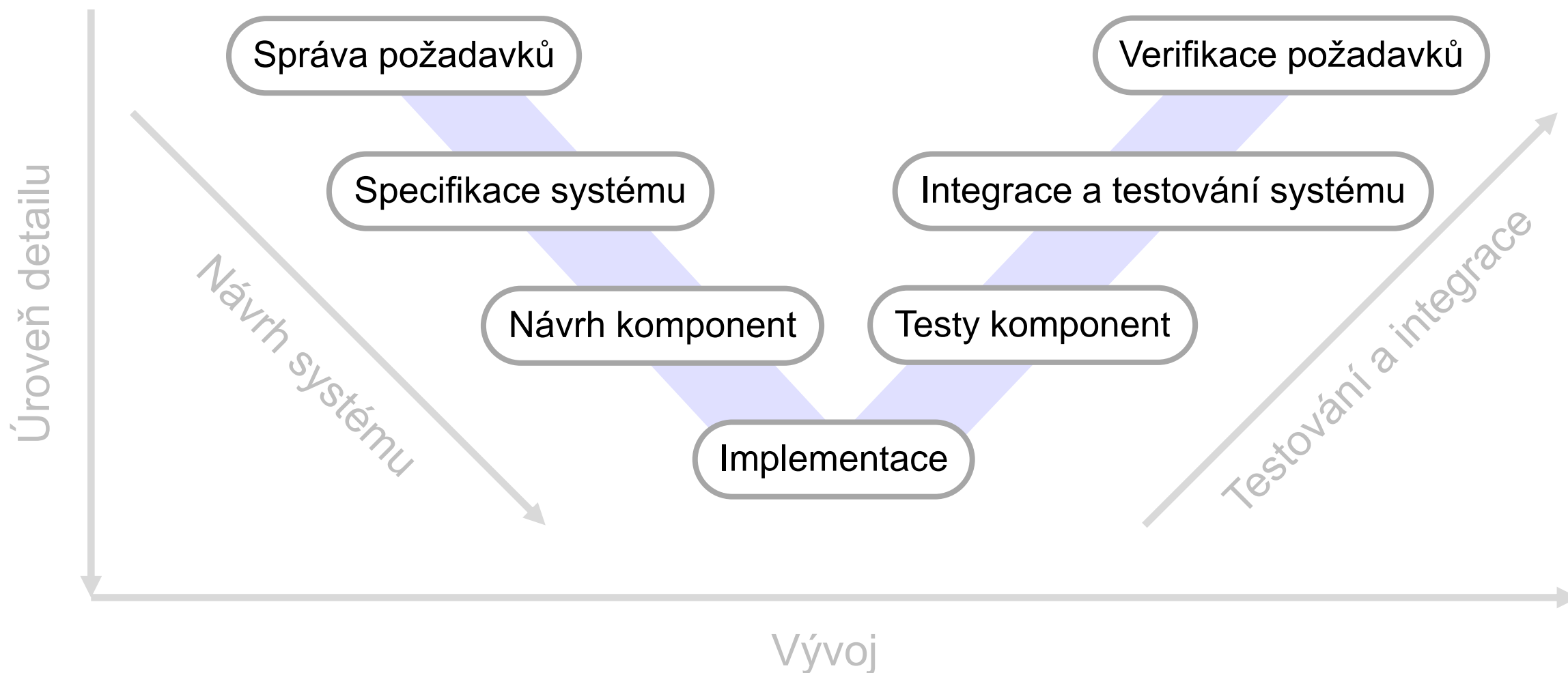
Modelování a simulace řídicích algoritmů a dynamických systémů (soustav)

Řídicí a rozhodovací logika

# MATLAB a Simulink pro návrh řídicích systémů

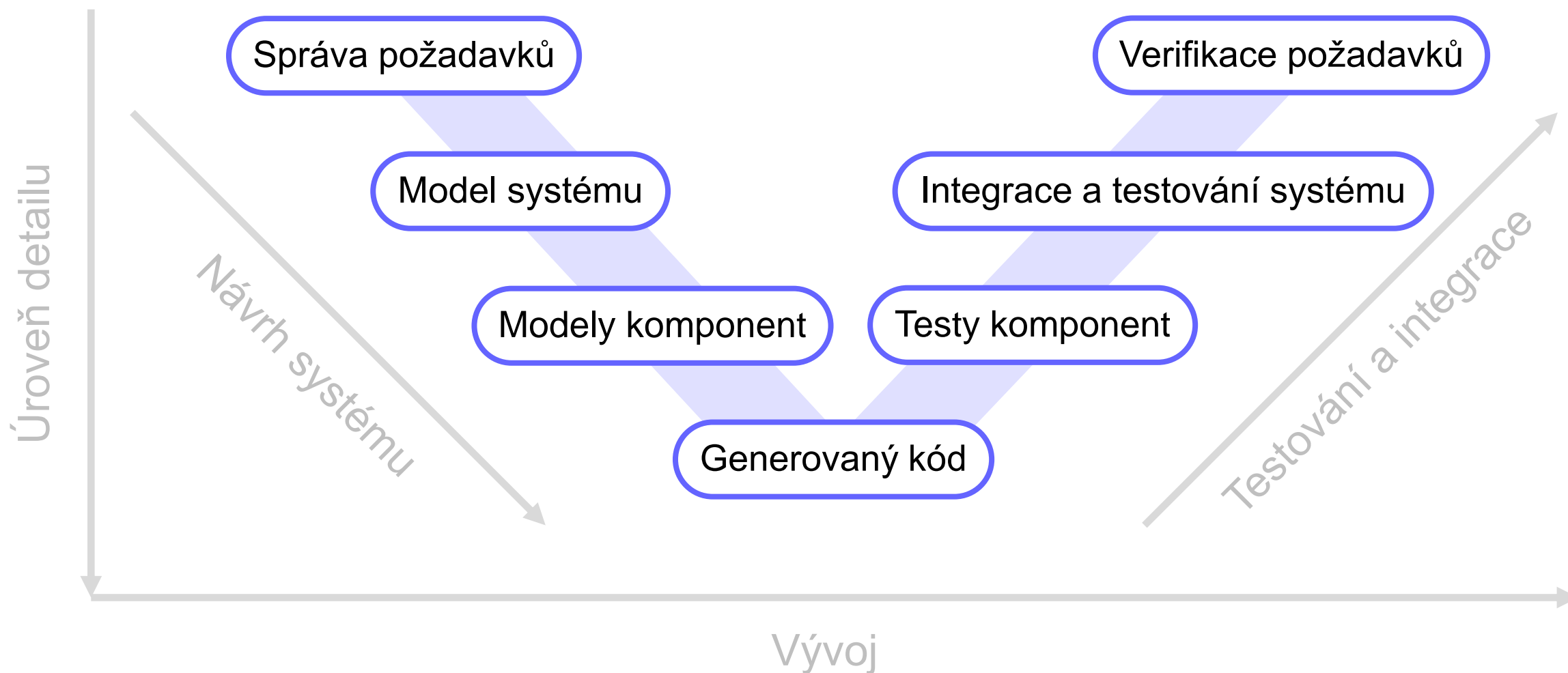


# Tradiční vývojový V-cyklus pro (řídící) systémy



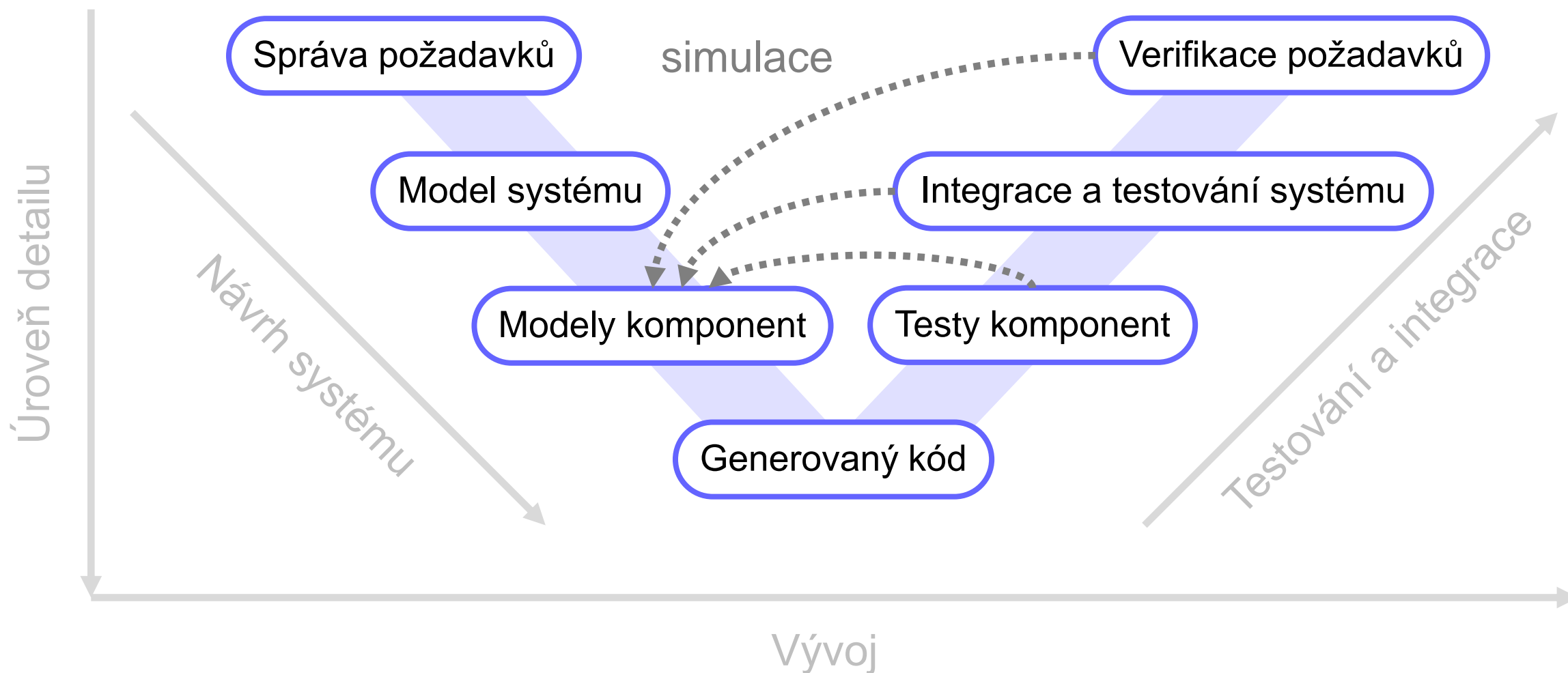
# Vývojový V-cyklus v pojetí MBD

Model-Based Design využívá návrh v podobě modelů a implementaci pomocí generování kódu



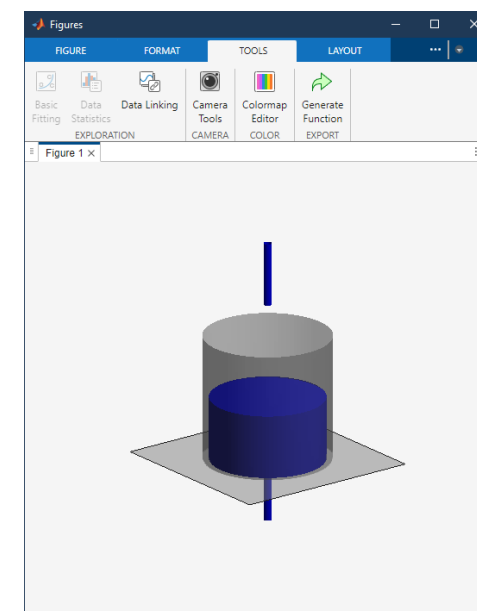
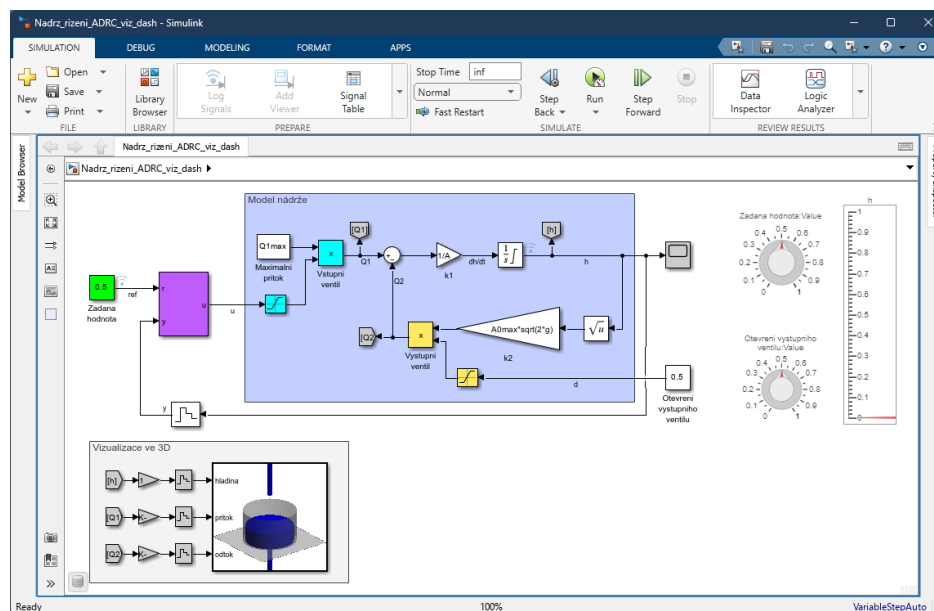
# Vývojový V-cyklus v pojetí MBD

Model-Based Design posouvá část testování (simulace modelů) do levé poloviny diagramu



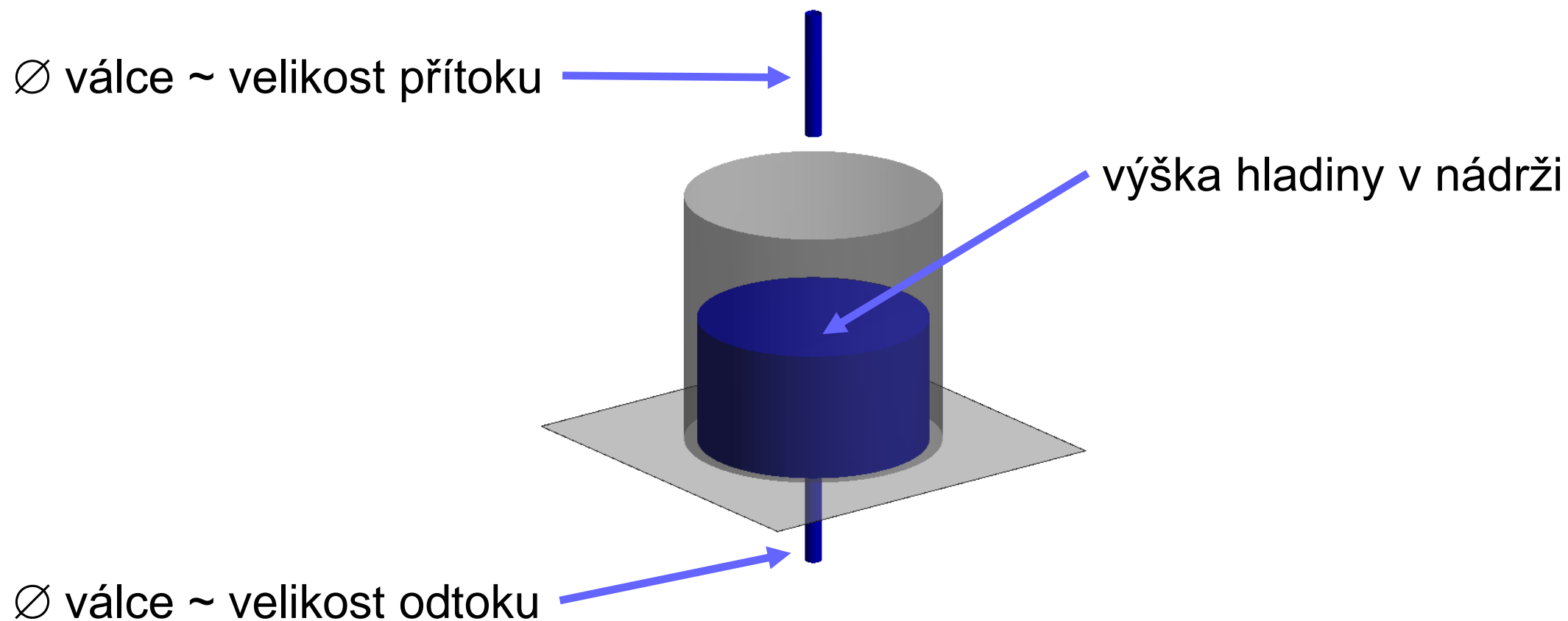
# Ukázka: Regulace hladiny v nádrži

- Nádrž s přítokem a výtokem řízená regulátorem
  - máme nádrž s řízeným přítokem Q1 a variabilním odtokem Q2
  - cíl: zajistit požadovanou výšku hladiny

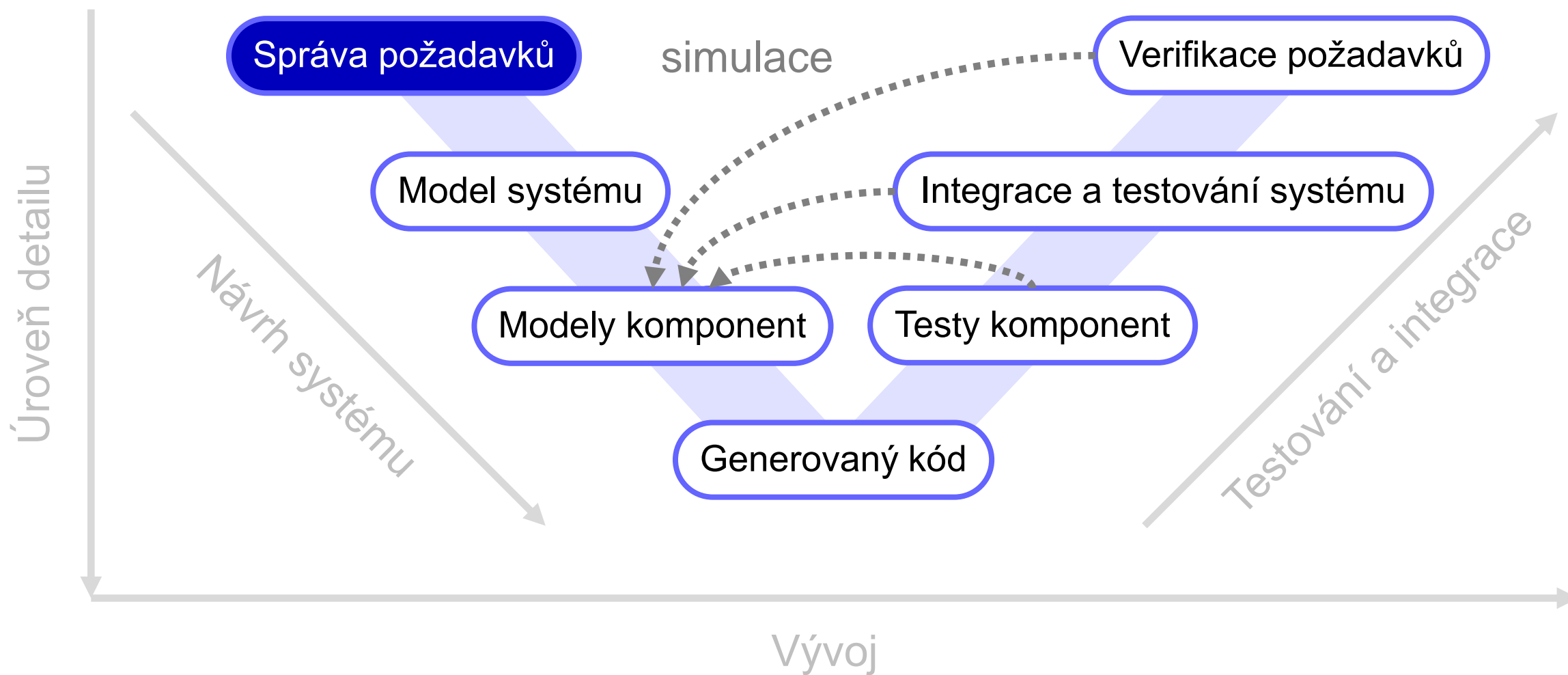


## Ukázka: 3D animace

- 3D animace vytvořena pomocí prostředků základního prostředí MATLAB
- Animace volána z modelu pomocí bloku *MATLAB Function*

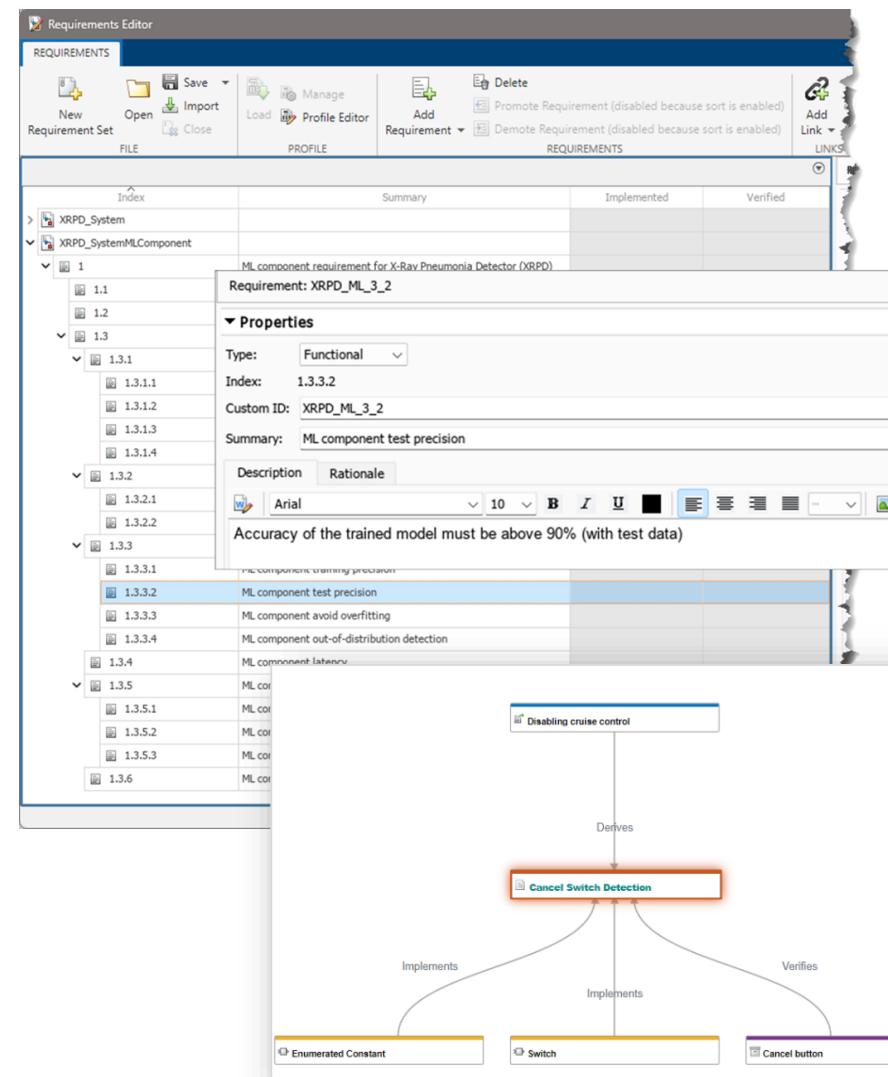


# Správa požadavků



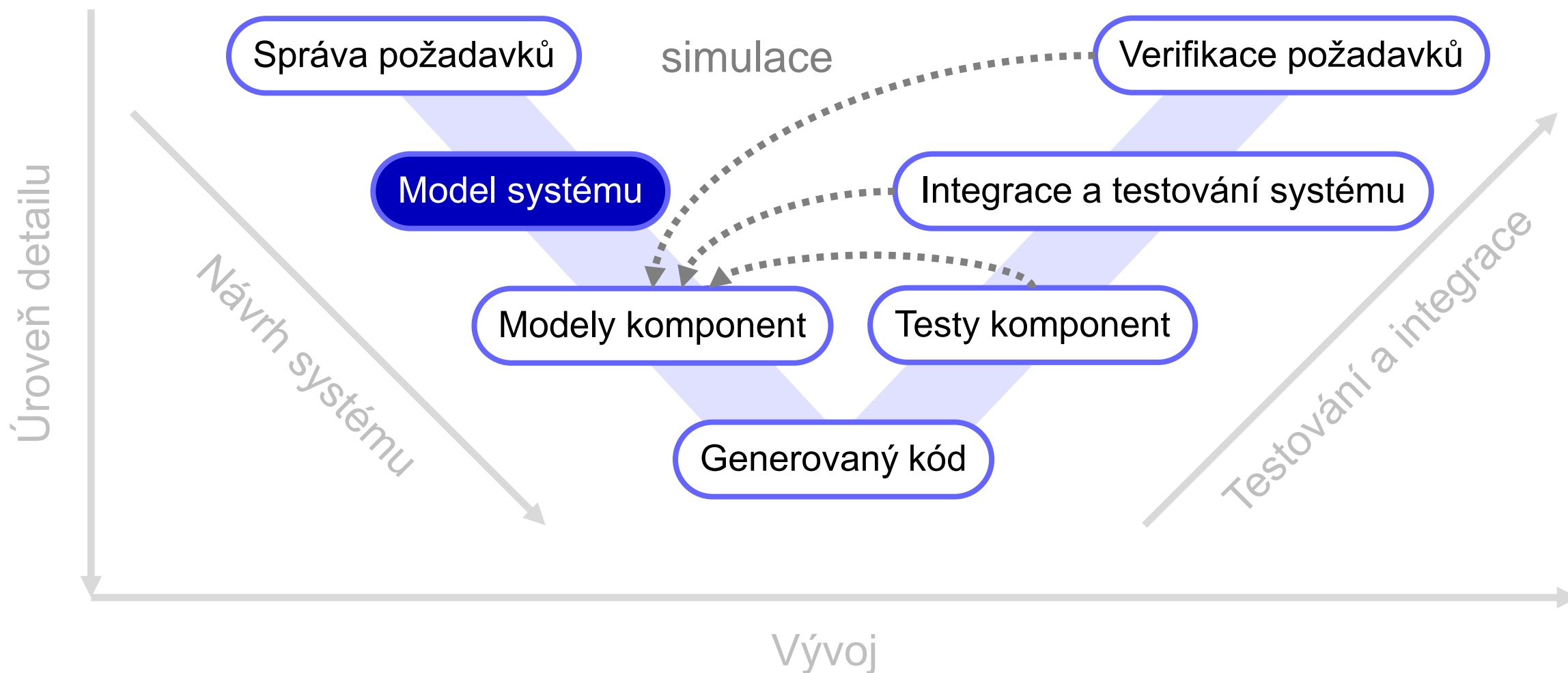
# Správa požadavků

- Požadavky na celkový návrh systému
- Požadavky specifické komponenty
- Requirements Toolbox
  - vytváření, propojení a trasování požadavků
  - Requirements Editor
  - Traceability Matrix, Traceability Diagrams
  - propojení/import: IBM Doors, MS Word, ReqIF
- Klíčové otázky
  - jsou všechny požadavky implementovány?
  - jak budou požadavky testovány?
  - lze vysvětlit chování modelu?



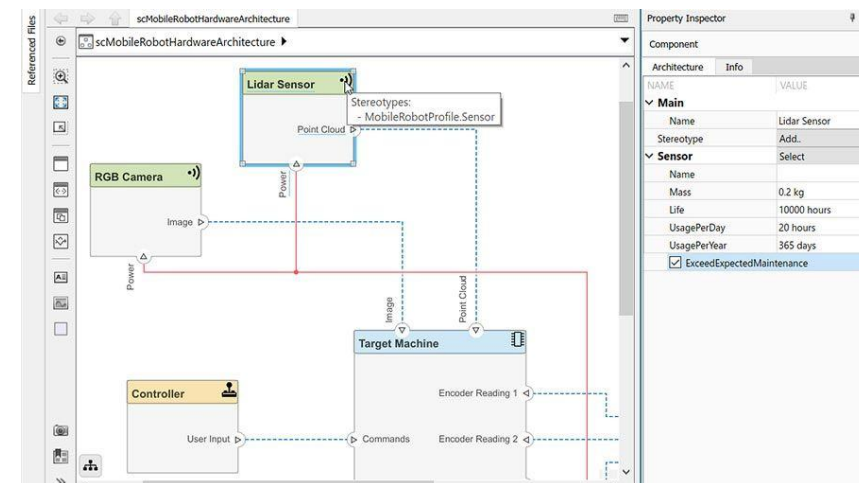
The screenshot shows the Requirements Editor interface. On the left, a tree view displays a hierarchy of requirements under 'XRPD\_SystemMLComponent'. A specific requirement, 'ML component test precision' (ID: XRPD\_ML\_3\_2), is selected. A detailed view of this requirement is shown on the right, including its properties (Type: Functional, Index: 1.3.3.2, Summary: ML component test precision) and a description: 'Accuracy of the trained model must be above 90% (with test data)'. Below the screenshot, a traceability diagram illustrates the relationships between requirements. A central requirement 'Cancel Switch Detection' is connected to 'Disabling cruise control' (Derives), 'Enumerated Constant' (Implements), 'Switch' (Implements), and 'Cancel button' (Verifies).

# Modelování systému ~ návrh architektury

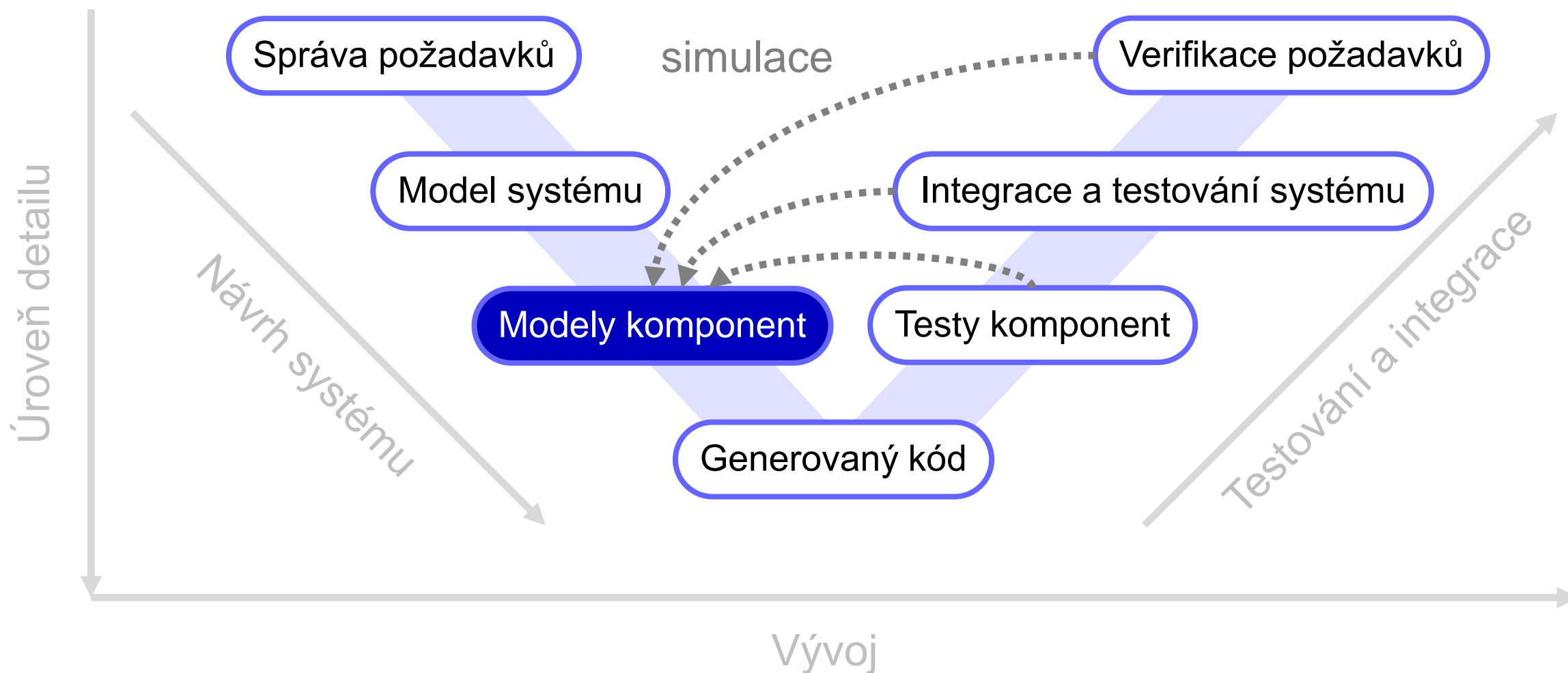


# Návrh architektury systému

- System Composer
  - nástroj pro návrh a dekompozici architektury systému
  - komponenty, rozhraní, stereotypy, pohledy, ...
  - propojení s požadavky
  - dále v návrhu lze provázat s modely komponent
- Simulink (bez nástroje System Composer)
  - architekturu představuje kořenová úroveň modelu systému
  - počáteční rozvržení architektury pomocí prázdných subsystémů



# Modelování komponent ~ návrh funkčnosti

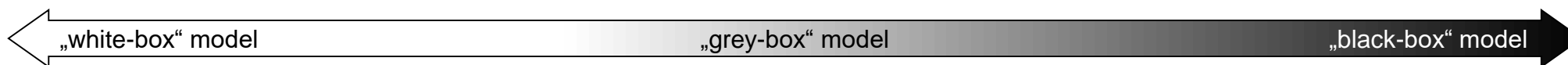


# Modelování soustav: Přístupy k modelování

- Pro různé situace jsou vhodné různé přístupy

**Fyzikální vztahy**

**Naměřená data**



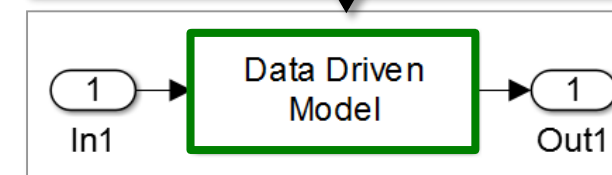
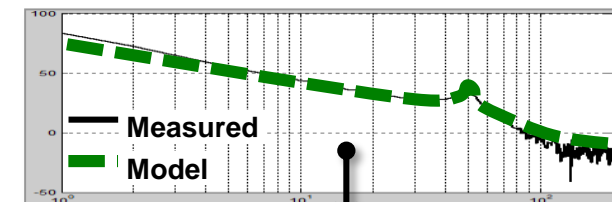
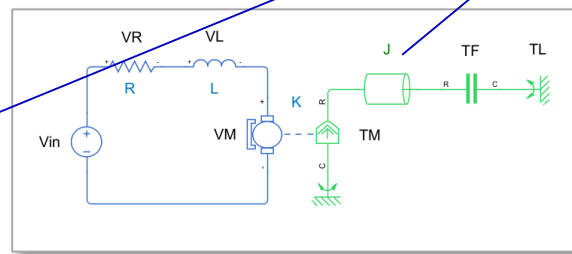
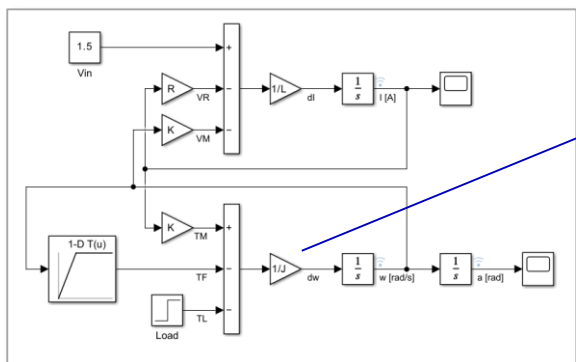
**Modelování rovnic**

**Ladění neznámých parametrů**

**Identifikace soustav**

**Fyzikální sítě**

$$L \frac{dI}{dt} = V_{in} - RI - K\omega \quad J \frac{d\omega}{dt} = KI - f(\omega) - T_L$$



# Modelování soustav: Prostředky

- Popis matematickými rovnicemi (Simulink)

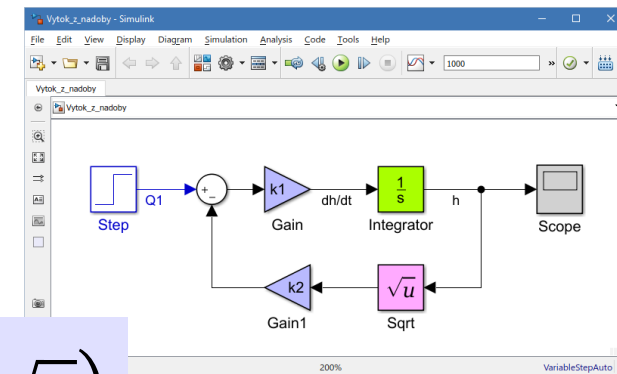
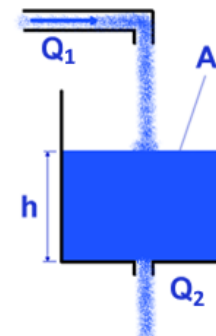
- matematicko-fyzikální analýza
- identifikace soustavy z naměřených dat

- Fyzikální modelování (Simscape)

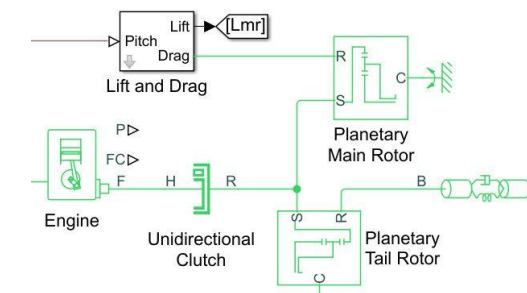
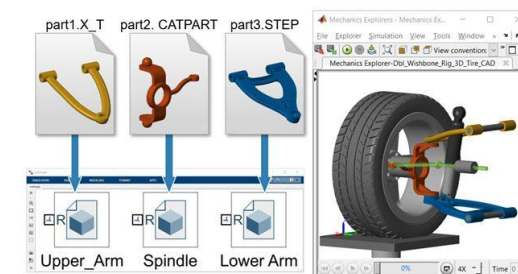
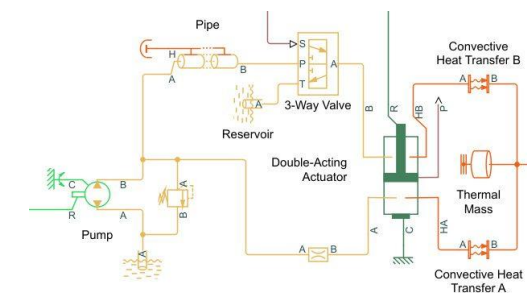
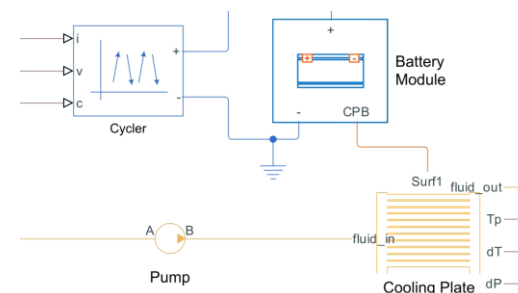
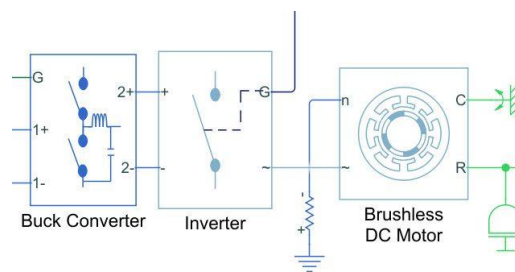
- elektrické a elektromechanické systémy
- baterie
- tekutinové systémy
- 3-D mechanika
- převodové systémy

- Aplikačně zaměřené

- automobily, letadla, drony, roboty



$$\frac{dh}{dt} = k_1(Q_1 - k_2\sqrt{h})$$



# Ukázka: Jak získat model nádrže?

- Odvození rovnice

$$\frac{dV}{dt} = Q_1 - Q_2 \quad V = A h \quad Q_2 = A_0 \sqrt{2gh}$$

$$A \frac{dh}{dt} = Q_1 - A_0 \sqrt{2gh}$$

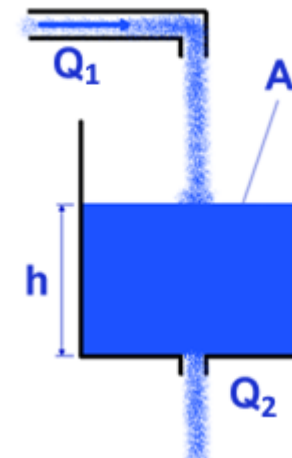
$$\frac{dh}{dt} = \frac{1}{A} (Q_1 - A_0 \sqrt{2gh}) \quad k_1 = \frac{1}{A} \quad k_2 = A_0 \sqrt{2g}$$

- Tvar vhodný pro modelování v Simulinku

$$\frac{dh}{dt} = k_1 (Q_1 - k_2 \sqrt{h})$$

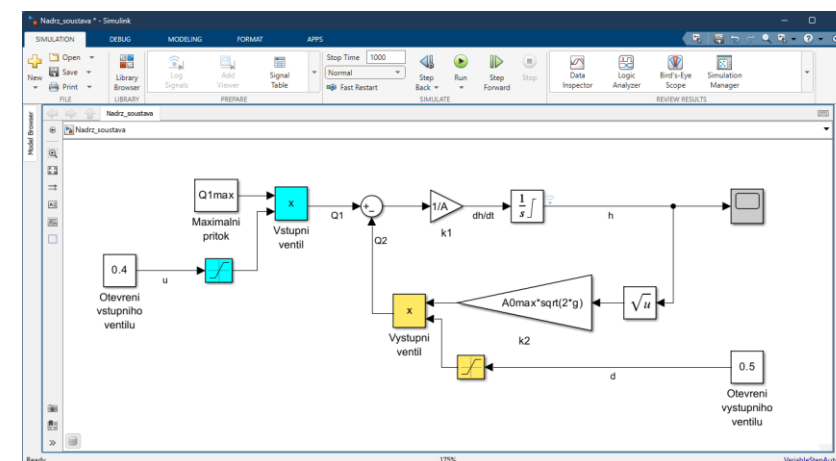
$$Q_1 = Q_{1\max} k_{vstup}$$

$$A_0 = A_{0\max} k_{výstup}$$



Hodnoty parametrů:

- $A = 0.2 \text{ m}^2$
- $A_{0\max} = 0.001 \text{ m}^2$
- $Q_{1\max} = 0.005 \text{ m}^3/\text{s}$

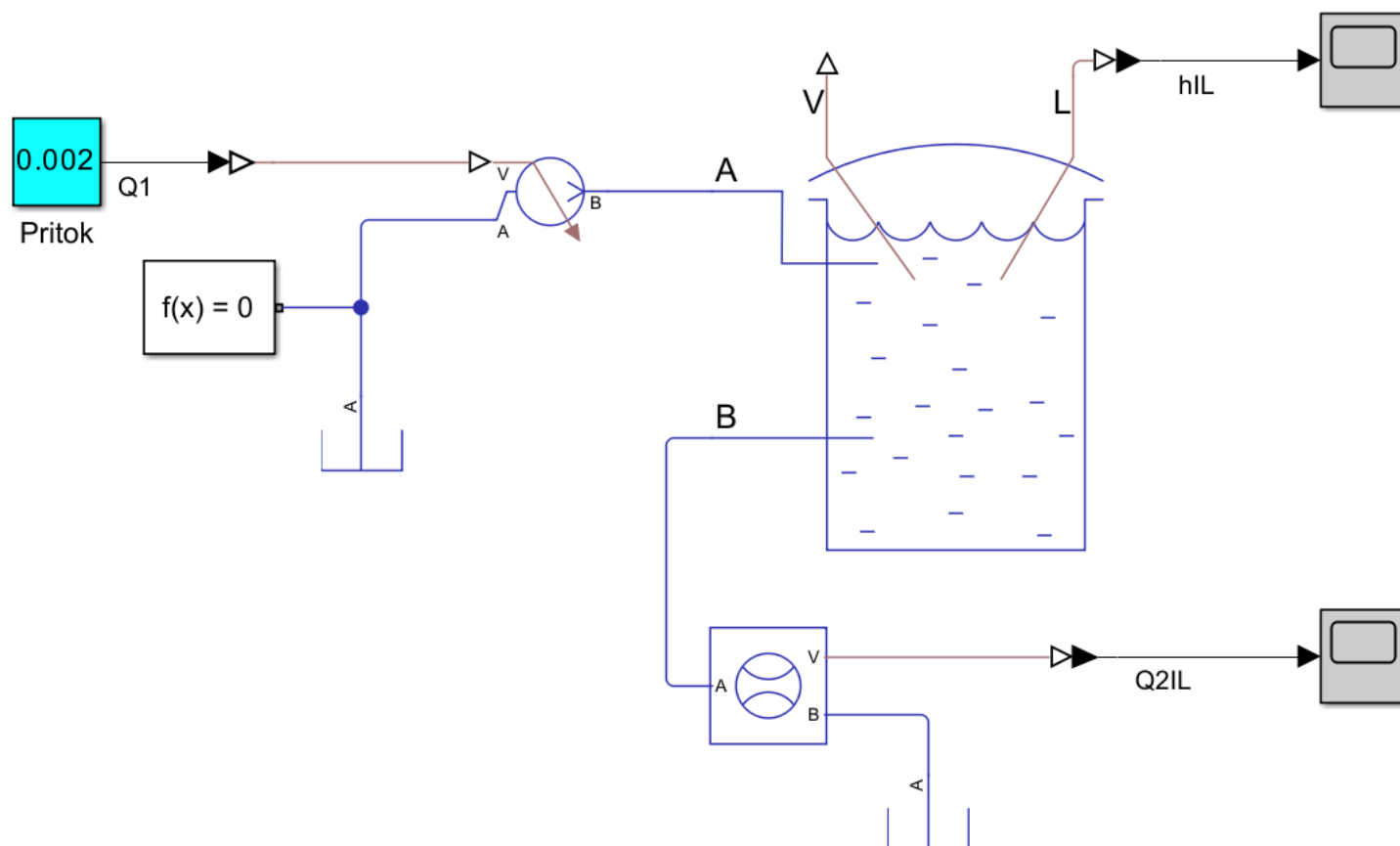


# Ukázka: Jak získat model nádrže?

- Nebo použít knihovny Simscape

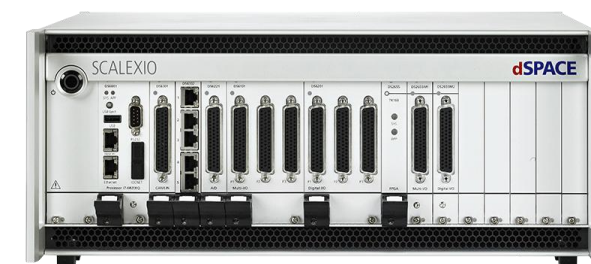
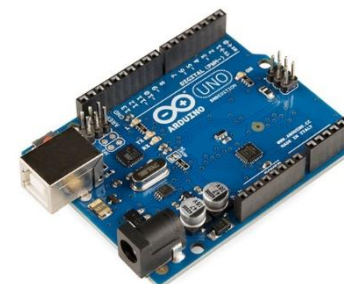
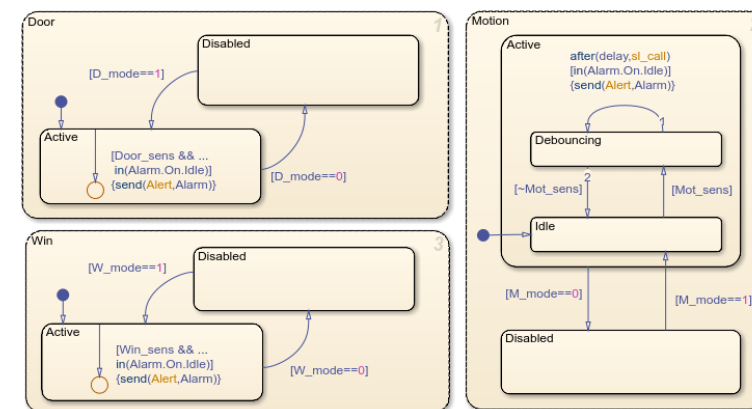
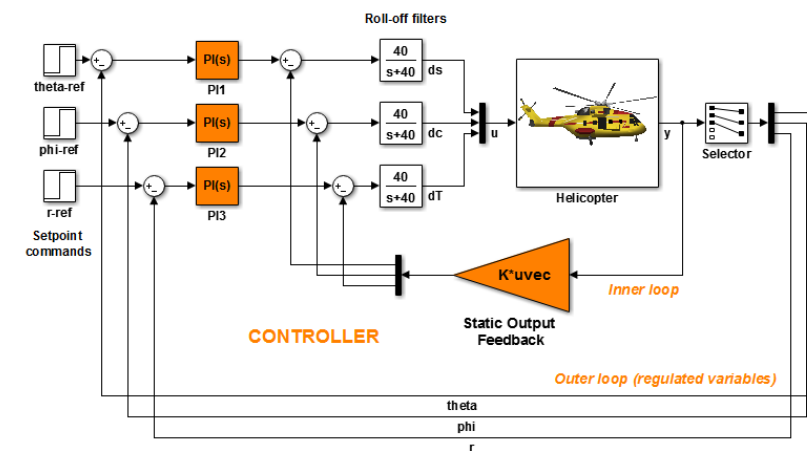
Hodnoty parametrů:

- $A = 0.2 \text{ m}^2$
- $A_{0\max} = 0.001 \text{ m}^2$
- $Q_{1\max} = 0.005 \text{ m}^3/\text{s}$



# Modelování algoritmů

- Řídicí systémy
- Logické a rozhodovací systémy
- Zpracování signálu a komunikace
- Zpracování obrazu a počítačové vidění
- a další ...
- Společná simulace soustav a algoritmů
- Generování kódu pro cílové platformy
  - C/C++, HDL, PLC (strukturovaný text), CUDA
  - simulace a testování v reálném čase, produkce



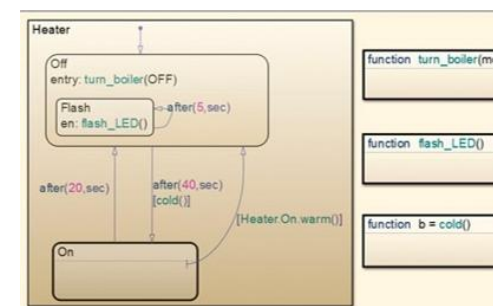
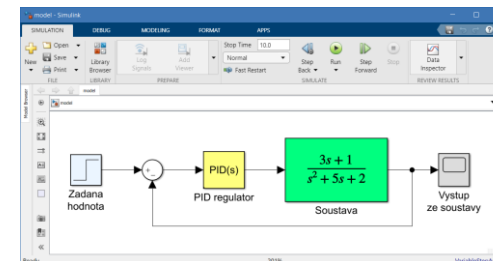
# Modelování algoritmů: Prostředky

Model v prostředí Simulink

Výpočetní a dynamické algoritmy  
**Simulink**

Stavové a logické algoritmy  
**Stateflow**

Kód  
**MATLAB / C**



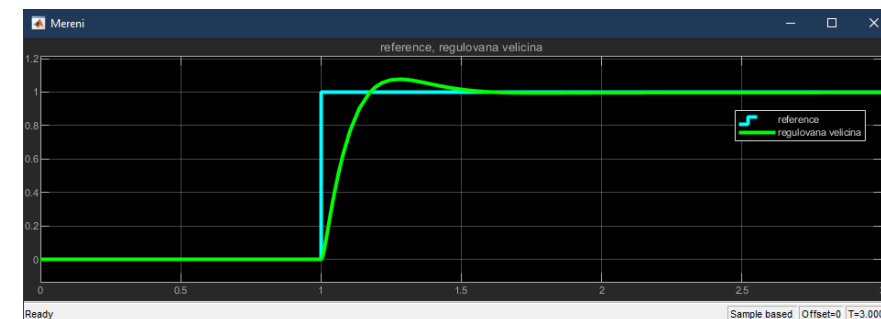
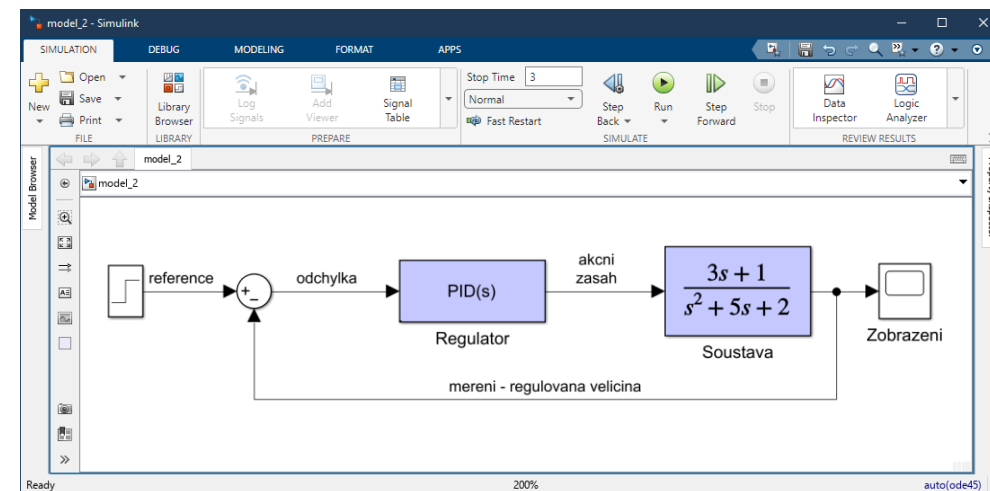
```

1 function [mean,stdev] = stats(vals)
2 % #codegen
3
4 % calculates a statistical mean and a standard
5 % deviation for the values in vals.
6
7 len = length(vals);
8 mean = avg(vals,len);
9 stdev = sqrt(sum((vals-avg(vals,len)).^2)/len);
10 coder.extrinsic('plot');
11 plot(vals,'-+');
12
13 function mean = avg(array,size)
14 mean = sum(array)/size;

```

# Návrh řídicích systémů v prostředí MATLAB a Simulink

- Propojení modelů soustav s regulátory
  - spojité a diskrétní prvky v jednom modelu
  - bohaté knihovny vstupních signálů
- Libovolná architektura řídicích systému
  - blok PID regulátoru v mnoha variantách
  - spojité, diskrétní, stavové regulátory
  - adaptivní a prediktivní řízení
- Nástroje pro ladění řídicích systémů



# Active Disturbance Rejection Control (ADRC)

- Řízení soustav s neznámou dynamikou a vnitřními a vnějšími poruchami
  - regulace s robustním potlačením poruch a minimálním překmitem

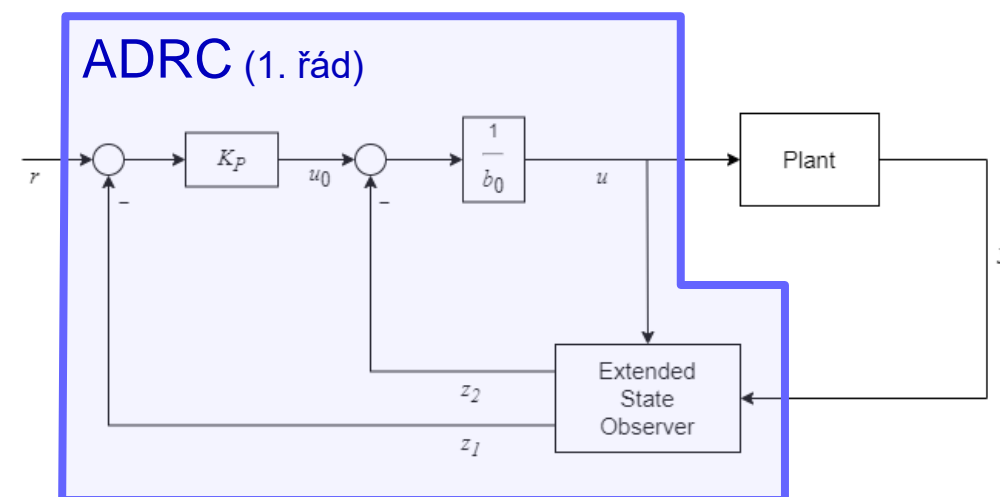
- Blok *Active Disturbance Rejection Control*

- Princip ADRC

- nevyžaduje model soustavy
- využívá jednoduchou aproximaci známé dynamiky

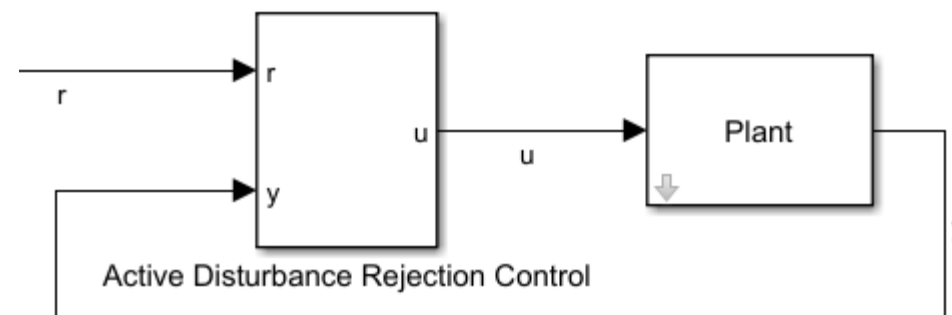
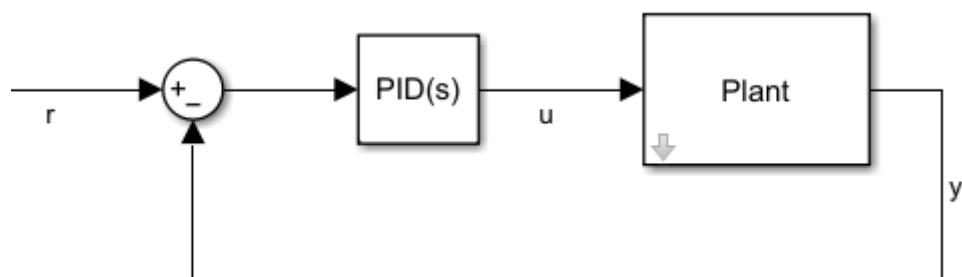
$$\dot{y}(t) = b_0 u(t) + f(t) \quad \text{nebo} \quad \ddot{y}(t) = b_0 u(t) + f(t)$$

- neznámá dynamika a poruchy  $f(t)$  modelovány jako rozšířený stav soustavy

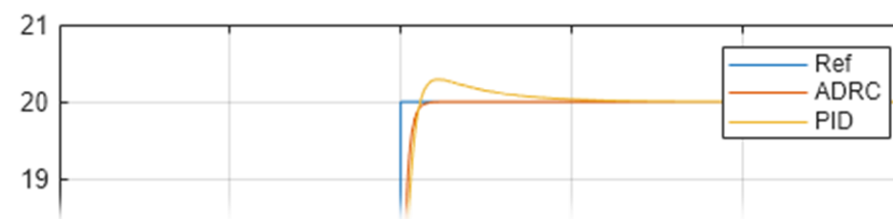


# Active Disturbance Rejection Control (ADRC)

- Může sloužit jako alternativa PID regulace
  - zejména pro výkonovou elektroniku a řízení motorů
  - obdobné výpočetní nároky jako PID

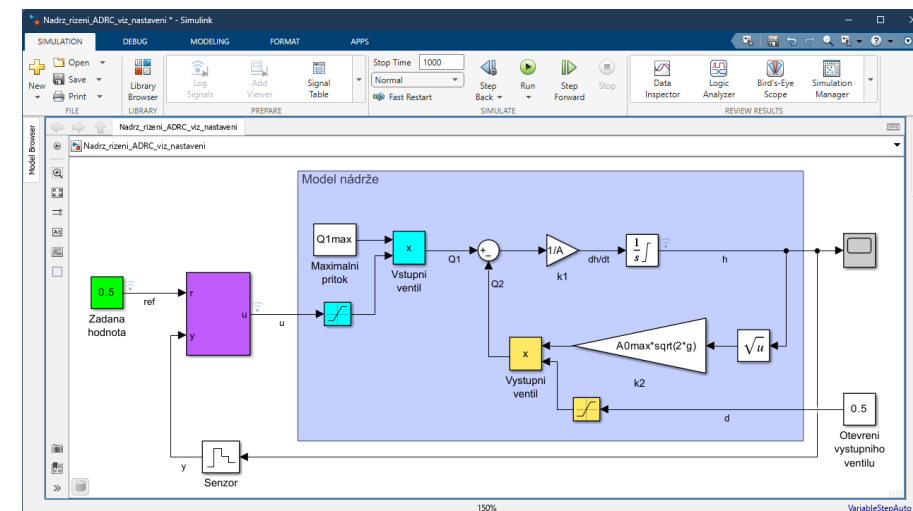


- Jednoduchý postup ladění
  - pro širokou škálu provozních podmínek



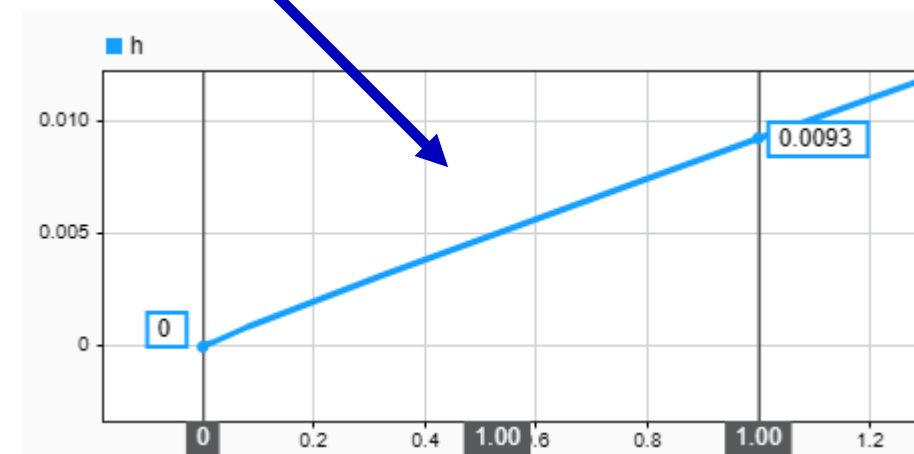
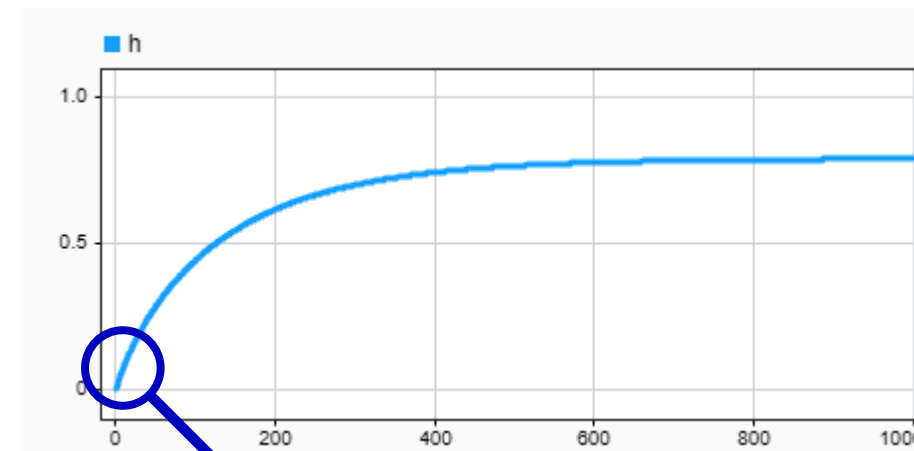
# Ukázka: Zapojení a nastavení ADRC regulace

- Zadána požadovaná hodnota výšky hladiny
  - konstantní hodnota  $ref = 0.5$
- Zapojena zpětnovazební regulace
  - blok *Active Disturbance Rejection Control*
  - není explicitní výpočet regulační odchylky
- Simulace chování senzoru výšky hladiny
- Nastavení parametrů ADRC regulátoru
  - dle postupu uvedeného v dokumentaci



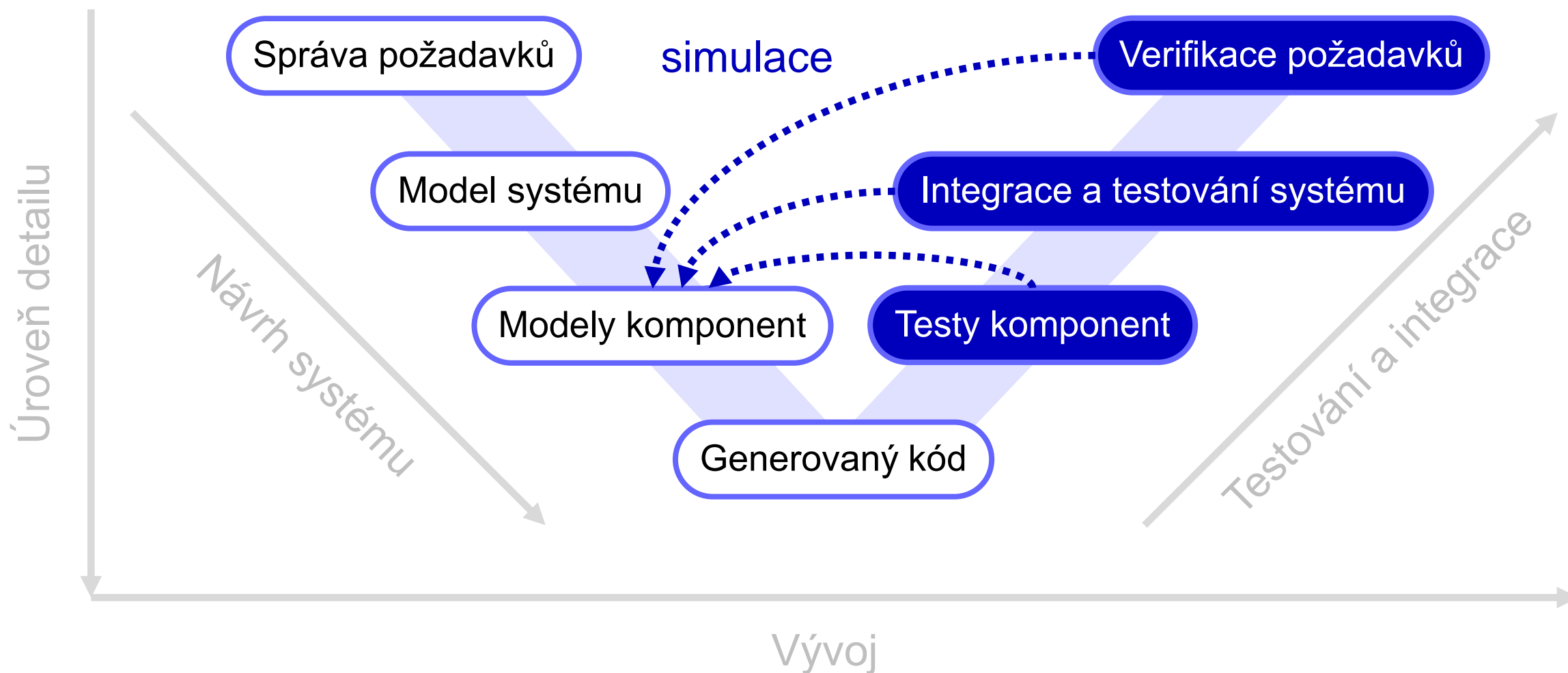
# Ladění ADRC regulátoru (1.řád)

- Určení kritického zesílení  $b_0$ 
  - experiment ~ přechodová odezva soustavy
  - proměření počátku odezvy
    - $a = \frac{\Delta y}{\Delta t}$ ,  $b_0 = \frac{a}{\Delta u}$
- Šířka frekvenčního pásma regulátoru
  - dle zadaných požadavků na chování systému
- Šířka frekvenčního pásma pozorovatele
  - 5 až 10 krát vyšší
- Hotovo ...



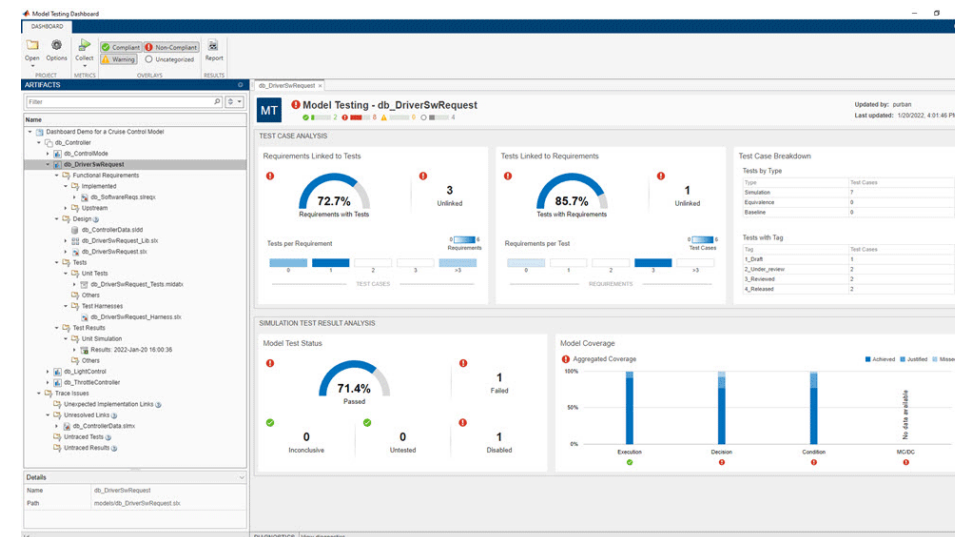
(pro model 2. řádu obdobně)

# Integrace a testování modelu systému a jeho komponent



# Testování a verifikace modelů

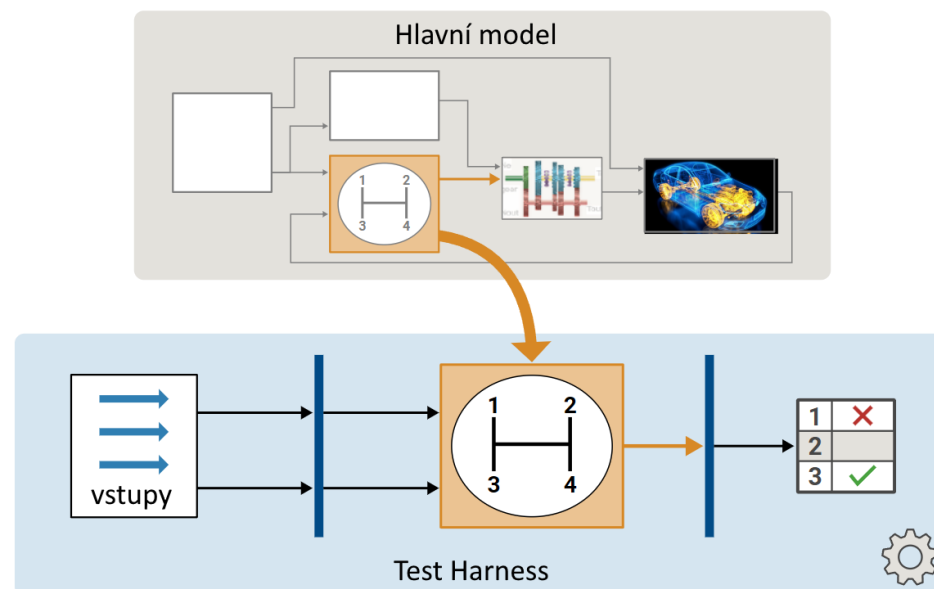
- Verifikace navrženého algoritmu v podobě simulačního modelu
  - ověření návrhu a chování algoritmu pomocí simulace (model-in-the-loop)
- Systematické testování prostřednictvím simulací
  - nástroje pro tvorbu a správu testů a verifikace požadavků (Test Harness, Test Manager)
  - analýza pokrytí testy (test coverage)
- Kontrola souladu s průmyslovými standardy
  - MISRA, ISO 26262, DO-178C, DO-331, ...
- Formální prokazování vlastností
- Verifikace požadavků na úrovni modelů
  - porovnání výsledků simulací s požadavky
  - provázání požadavků, modelů a testů





# Test Harness a Test Manager

- Test Harness = speciální testovací model
  - uložený s testovaným modelem nebo externě
  - lze aplikovat na celý model, vybraný subsystém v modelu, atd.

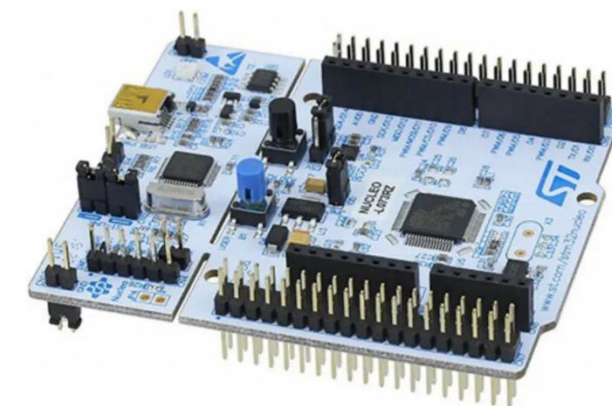


- Test Manager

– grafické rozhraní pro tvorbu, správu, spouštění a hodnocení výsledků testů

# Testování v reálném čase

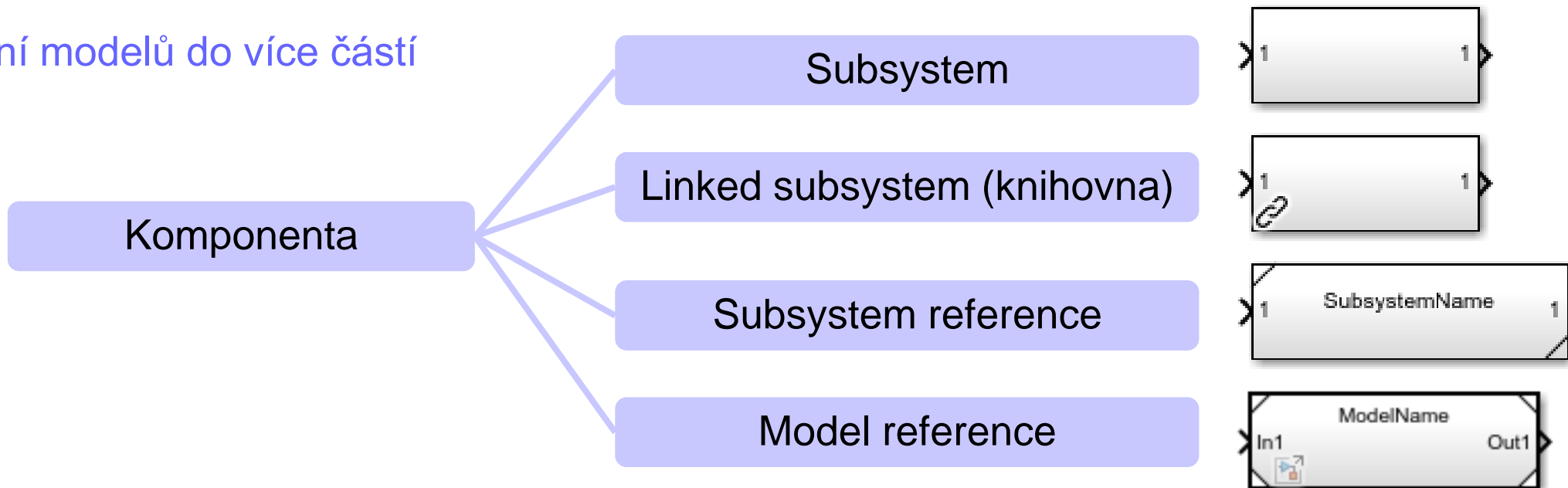
- Rapid Control Prototyping (RCP)
  - připojení modelu algoritmu v Simulinku k reálnému zařízení
  - ověření chování navrženého systému s reálnou soustavou
  - ladění parametrů
- Jaký použít hardware?
- Real-time simulátor ~ tradiční RCP
  - cíl: ověřit funkčnost algoritmu v reálném prostředí
  - výkon, flexibilita
- Vývojová embedded platforma ~ on-target RCP
  - cíl: ověřit praktickou realizovatelnost algoritmu pro výrobu



# Integrace modelů komponent

- Komponentizace

- členění modelů do více částí



- Integrace modelů komponent

- zapojení modelů komponent do modelu systému

- propojení modelů až do kořenové úrovně nebo do modelu architektury

# Ukázka: Rozdělení modelu systému do komponent

- Části modelu

- řídicí systém ADRC
- soustava
- vstupní signály

- Typy komponent v modelu

- ADRC = referencovaný model

- když vyžadujeme možnost samostatného nastavení a simulace

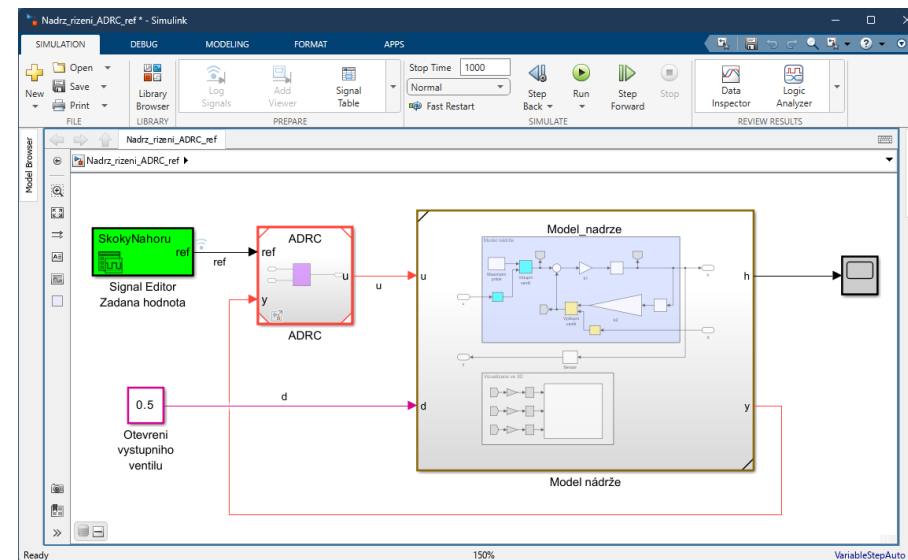
- Soustava = referencovaný subsystém

- když nevyžadujeme možnost samostatného nastavení a simulace, ale chceme často upravovat

- Alternativa: Soustava = knihovna

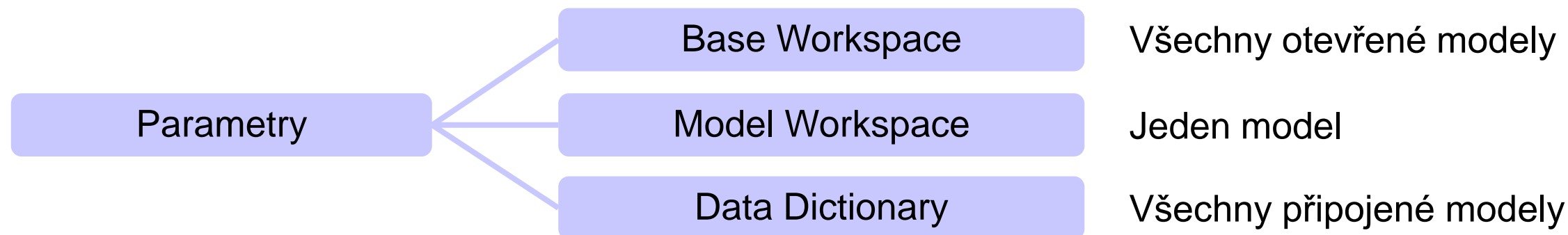
- nevyžadujeme možnost samostatného nastavení a simulace

- pro opakovaně využitelnou funkčnost menší komponenty, která se zřídka upravuje



# Správa dat v modelech

- Kde mohou být umístěny parametry

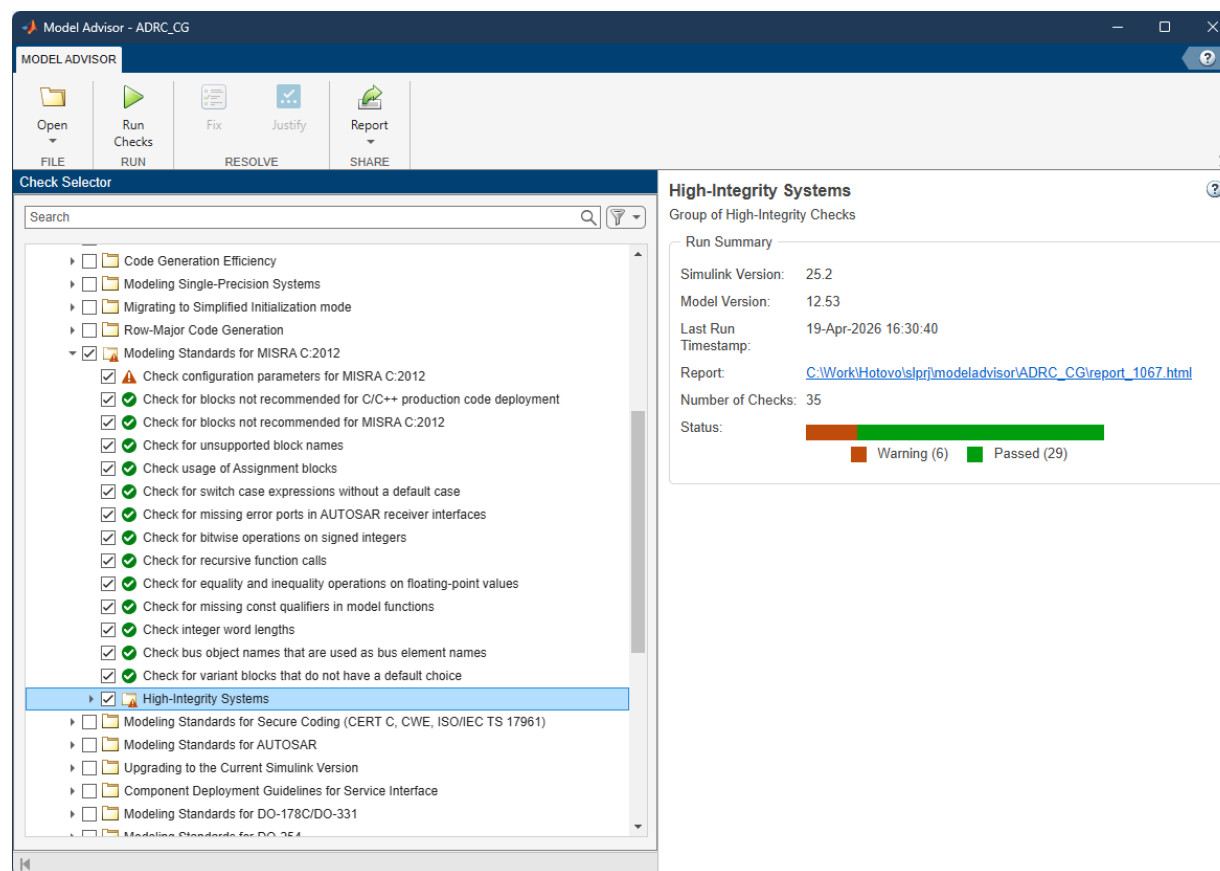


- Jak mohou být uspořádány signály

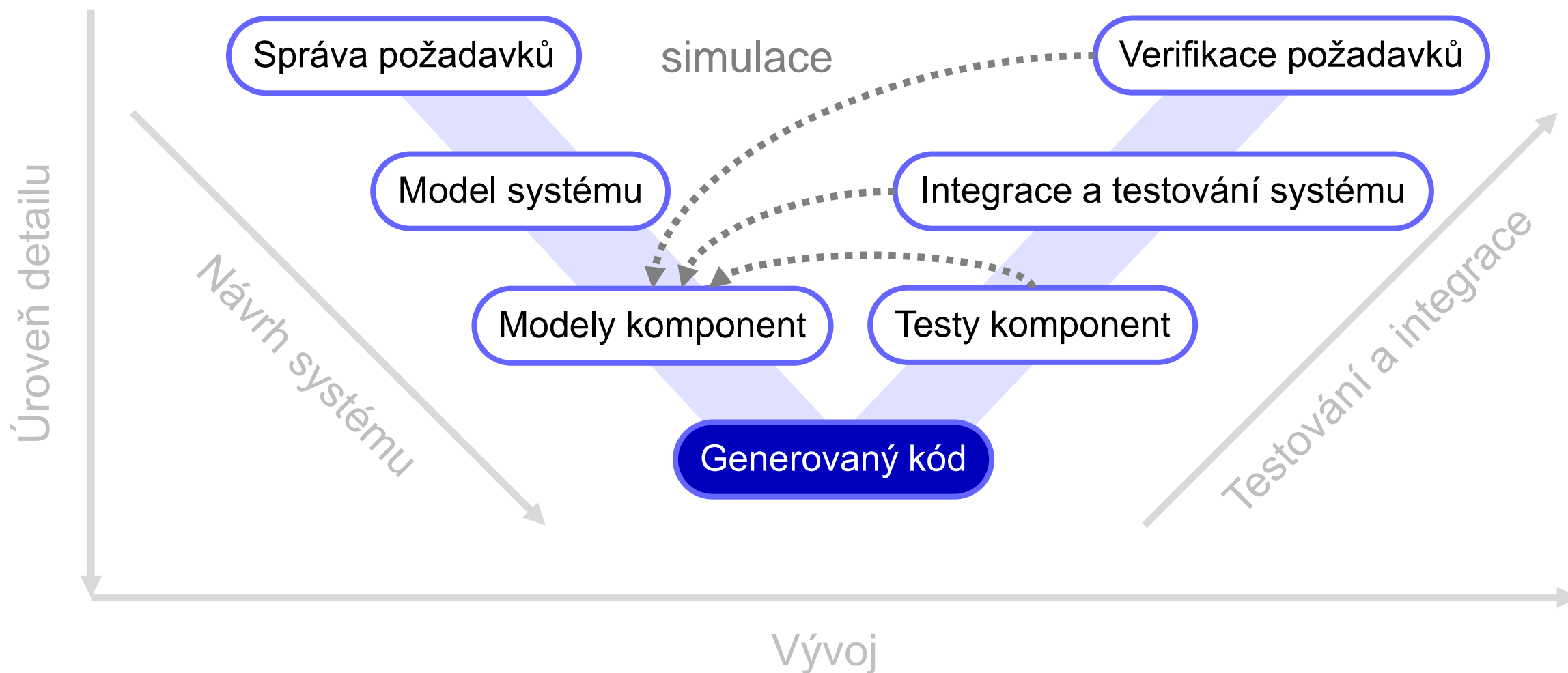


# Ukázka: Kontrola souladu se standardy

- Kontrola souladu ADRC se standardem MISRA-C

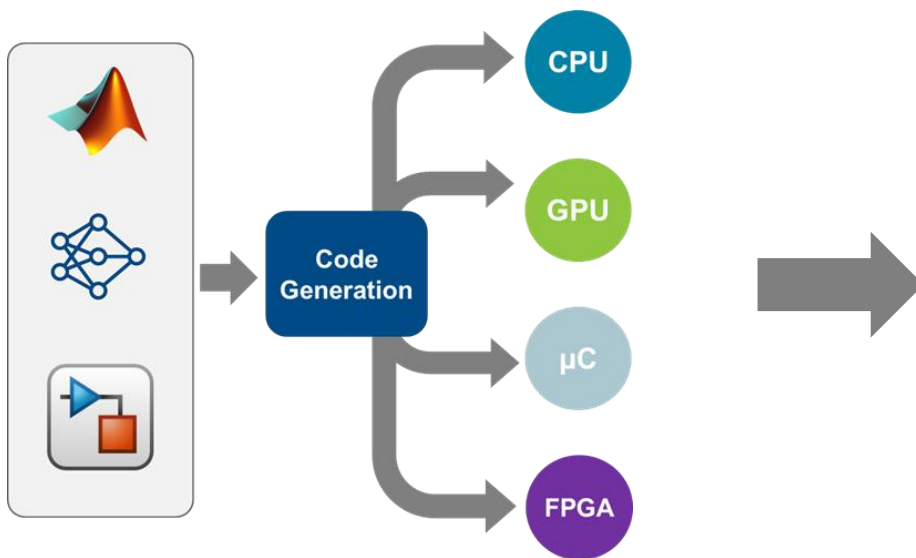
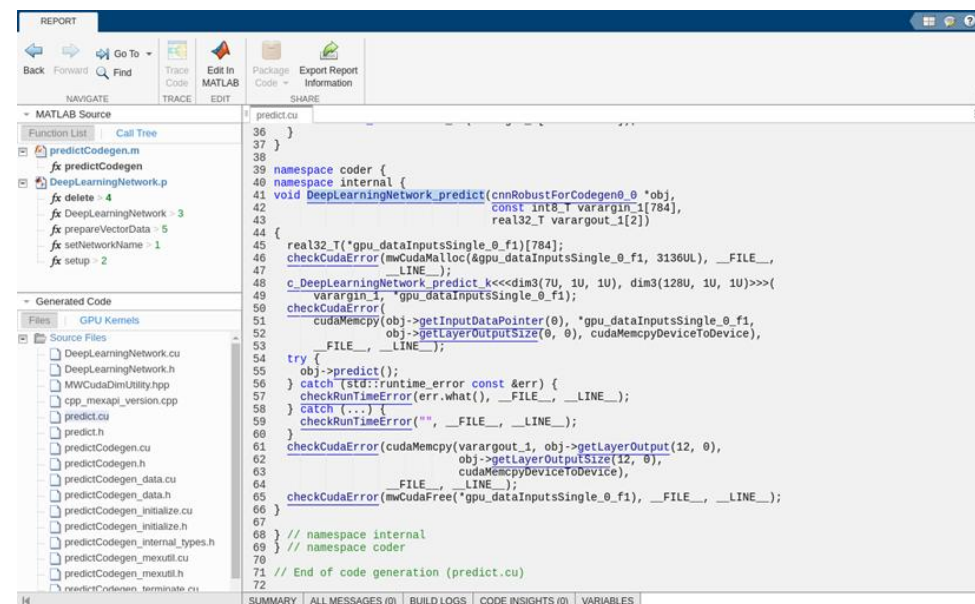


# Automatické generování kódu



# Implementace modelu

- Příprava modelu pro implementaci na cílovou platformu
  - Embedded Coder Quick Start
  - nastavení vzhledu generovaného kódu a optimalizace pro zvolenou cílovou platformu
- Automatické generování kódu

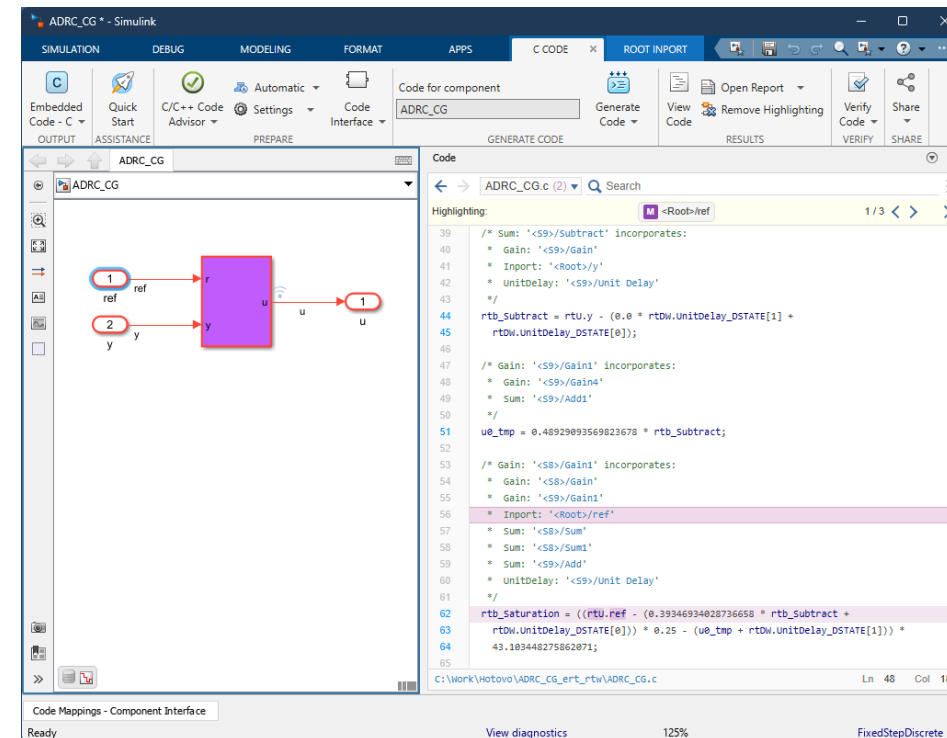
The screenshot shows the MATLAB Embedded Coder report interface. The left pane displays the "MATLAB Source" and "Generated Code" sections. The right pane shows the generated C++ code for a GPU target, including namespace definitions, memory allocation, and CUDA kernel calls.

```

36 }
37 }
38 }
39 namespace coder {
40 namespace internal {
41 void DeepLearningNetwork_predict(cnnRobustForCodegen_0 *obj,
42                                const int8_T varargin_1[784],
43                                real32_T varargin_2[2])
44 {
45     real32_T(*gpu_dataInputsSingle_0_f1)[784];
46     checkCudaError(mxCudaMalloc(&gpu_dataInputsSingle_0_f1, 3136UL), __FILE__,
47                   __LINE__);
48     c_DeepLearningNetwork_predict_k<<<dim3(7U, 1U, 1U), dim3(128U, 1U, 1U)>>(
49         varargin_1, *gpu_dataInputsSingle_0_f1);
50     checkCudaError(cudaMemcpy(obj->getInputDataPointer(0), *gpu_dataInputsSingle_0_f1,
51                             obj->getLayerOutputSize(0, 0), cudaMemcpyDeviceToDevice),
52                   __FILE__, __LINE__);
53 }
54 try {
55     obj->predict();
56 } catch (std::runtime_error const &err) {
57     checkRuntimeError(err.what(), __FILE__, __LINE__);
58 } catch (...) {
59     checkRuntimeError("", __FILE__, __LINE__);
60 }
61 checkCudaError(cudaMemcpy(varargin_2, obj->getLayerOutput(12, 0),
62                         obj->getLayerOutputSize(12, 0),
63                         cudaMemcpyDeviceToDevice),
64               __FILE__, __LINE__);
65 checkCudaError(mxCudaFree(*gpu_dataInputsSingle_0_f1), __FILE__, __LINE__);
66 }
67 }
68 // namespace internal
69 // namespace coder
70 }
71 // End of code generation (predict.cu)
72
  
```

# Ukázka: Generování kódu

- Model řídicího systému
  - bez modelu soustavy a testovacích vstupů
- Embedded Coder
  - nastavení parametrů pro generování kódu
- Trasování z modelu do kódu a opačně



The screenshot displays the MATLAB Embedded Coder environment. On the left, a Simulink block diagram for 'ADRC.CG' is shown, featuring a control block with two inputs labeled 'ref' and 'y', and one output labeled 'u'. The 'ref' input is connected to a summing junction, and the 'y' input is connected to a feedback path. The output 'u' is fed back to the 'y' input.

On the right, the generated C code is visible. The code includes comments for each block in the diagram and the corresponding mathematical operations. Key lines of code include:

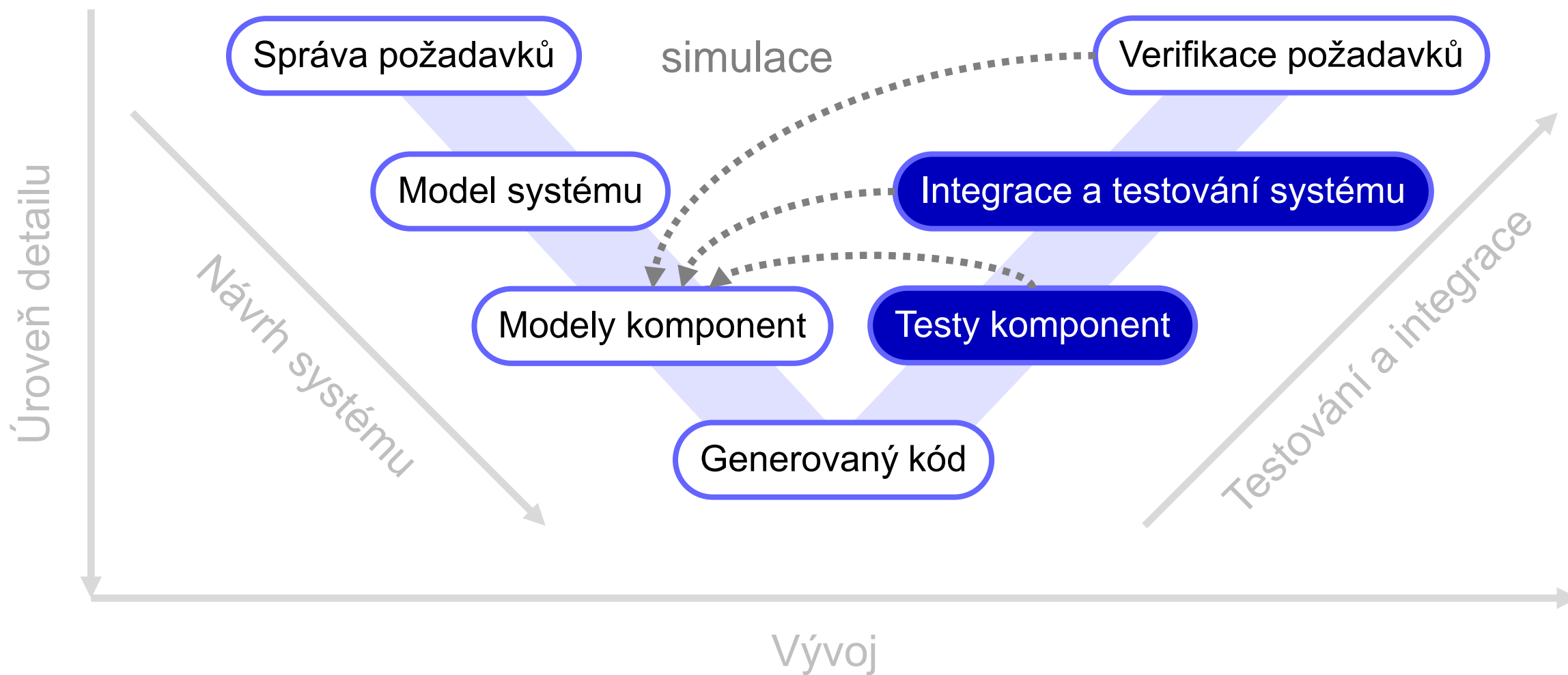
```

39 /* Sum: '<S9>/Subtract' incorporates:
40 * gain: '<S9>/gain'
41 * Inport: '<Root>/y'
42 * UnitDelay: '<S9>/Unit Delay'
43 */
44 rtb_Subtract = rtu.y - (0.0 * rtDW.UnitDelay_DSTATE[1] +
45   rtDW.UnitDelay_DSTATE[0]);
46
47 /* Gain: '<S9>/gain1' incorporates:
48 * gain: '<S9>/gain4'
49 * Sum: '<S9>/Add1'
50 */
51 u0_tmp = 0.48929893569923678 * rtb_Subtract;
52
53 /* Gain: '<S8>/gain1' incorporates:
54 * gain: '<S9>/gain'
55 * gain: '<S9>/gain1'
56 * Inport: '<Root>/ref'
57 * Sum: '<S8>/Sum'
58 * Sum: '<S8>/Sum1'
59 * Sum: '<S9>/Add'
60 * UnitDelay: '<S9>/Unit Delay'
61 */
62 rtb_Saturation = ((rtu.ref - (0.39346934828736658 * rtb_Subtract +
63   rtDW.UnitDelay_DSTATE[0])) * 0.25 - (u0_tmp + rtDW.UnitDelay_DSTATE[1])) *
64   43.103448275862071;
65

```

The status bar at the bottom indicates 'Code Mappings - Component Interface' and 'Ready'.

# Testování a integrace kódu

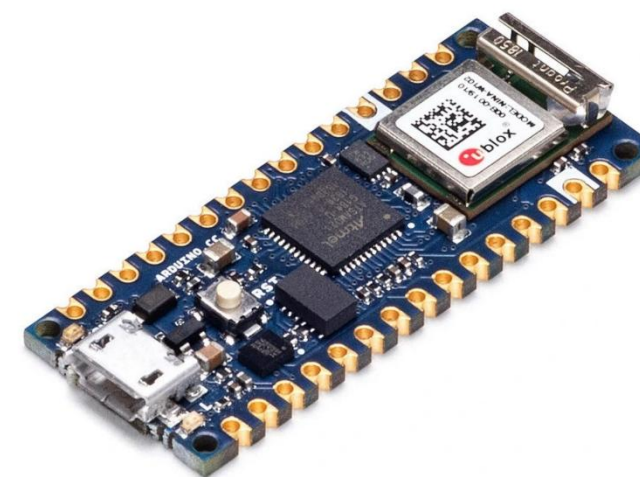
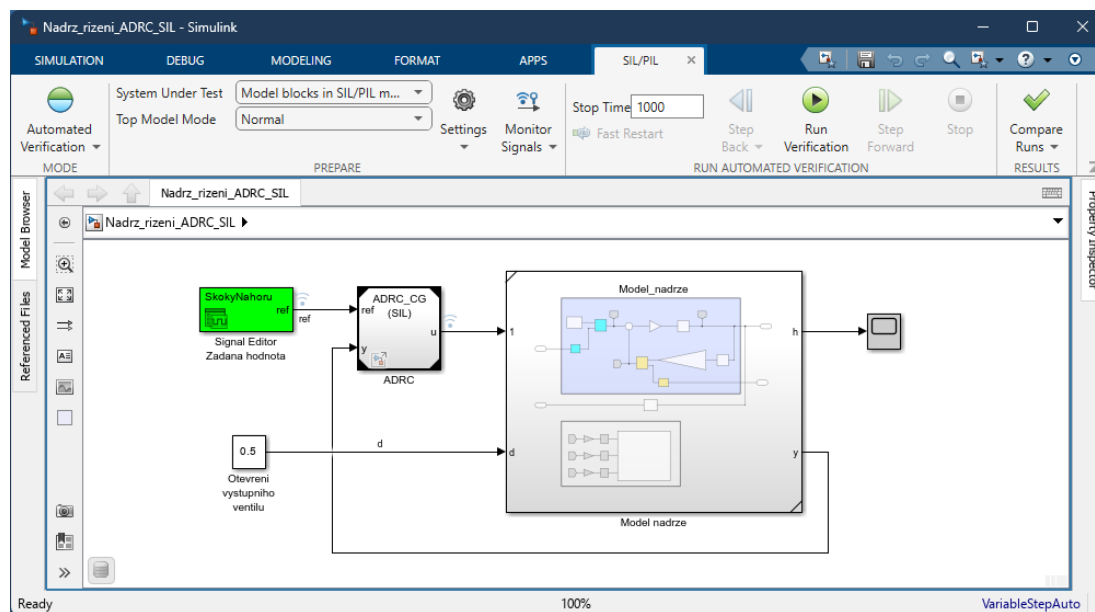


# Testování a integrace kódu

- Testování implementovaného algoritmu
  - ověření chování algoritmu v implementované podobě
  - zavedení generovaného kódu zpět do Simulinku a provedení testů: *software-in-the-loop* (SIL)
  - propojení Simulinku s cílovou platformou, testování na cílovém HW: *processor-in-the-loop* (PIL)
- Integrace kódu
  - zapojení generovaného kódu do širšího rámce
  - propojení s externími částmi kódu
- Simulink jako platforma pro integraci
  - integrace může být řešena současně s předchozí fází implementace
  - bloky v prostředí Simulink pro integraci C/C++ kódu
  - výsledný kód obsahuje generované části společně s integrovanými

# Ukázka: SIL a PIL simulace

- Model řídicího systému v implementované podobě
- Provedení simulačních testů s generovaným kódem
  - software-in-the-loop
  - processor-in-the-loop (Arduino Nano 33 IoT)



# Verifikace kódu

- Verifikace implementovaného algoritmu
  - zásadní zejména pro bezpečnostně kritické aplikace
- Formální analýza kódu
  - nástroje Polyspace
  - prokázání nepřítomnosti run-time chyb
  - využívá formální metody (= bez spouštění)

**Green: reliable**  
safe pointer access

**Red: faulty**  
out of bounds error

**Gray: dead**  
unreachable code

**Orange: unproven**  
may be unsafe for some conditions

**Purple: violation**  
MISRA-C/C++ or JSF++ code rules

**Range data**  
tool tip

```

static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }

    i = get_bus_status();

    if (i >= 0) {
        *(p - i) = 10;
    }
}

```

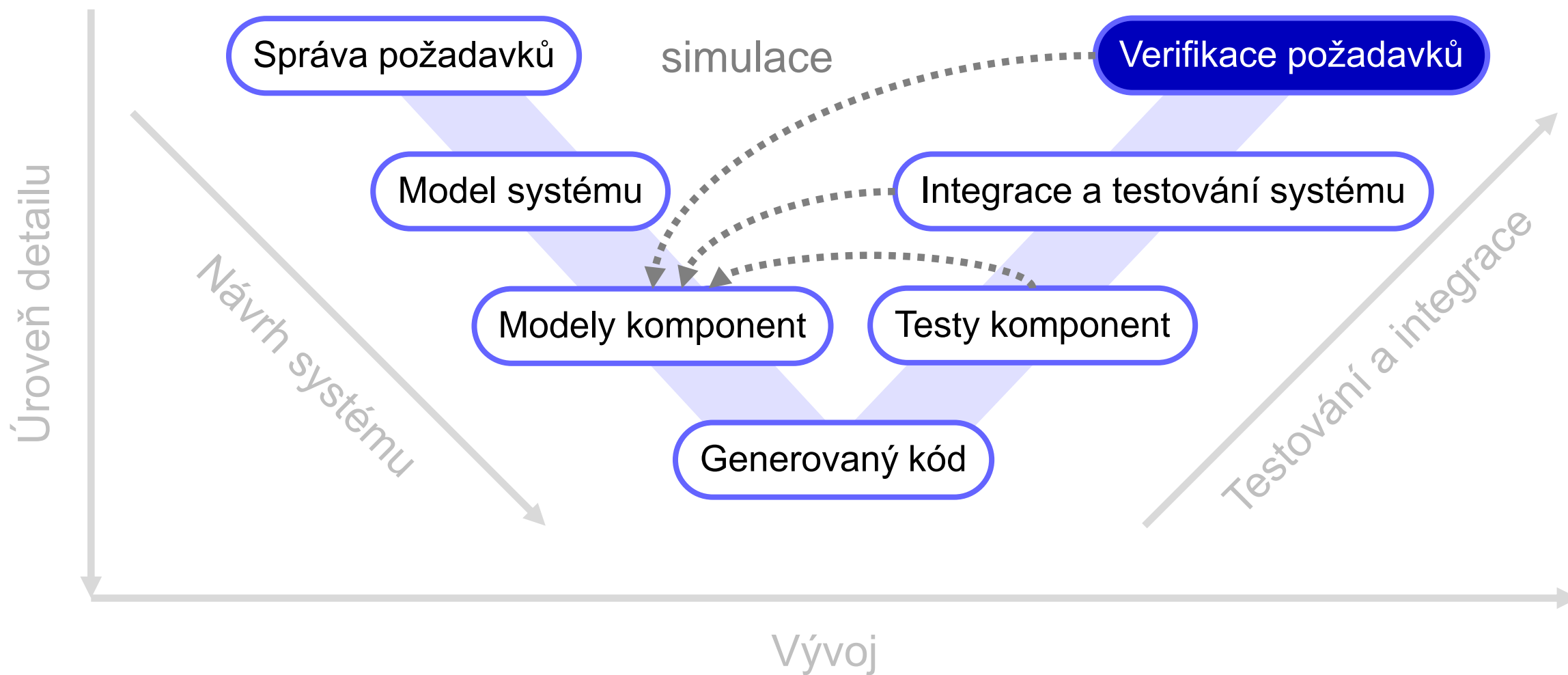
variable 'i' (int32): [0 .. 99]  
assignment of 'i' (int32): [1 .. 100]

# Testování vyrobeného zařízení

- Testování v reálném čase
  - *hardware-in-the-loop* (HIL)
- Typické uspořádání
  - vyvinuté reálné zařízení připojeno k simulačnímu modelu soustavy
  - simulační model spuštěn v reálném čase
  - ~ chová se jako reálný systém
- Real-Time simulátor
  - periférie odpovídají připojení reálného systému
  - ~ připojuje se jako reálný systém

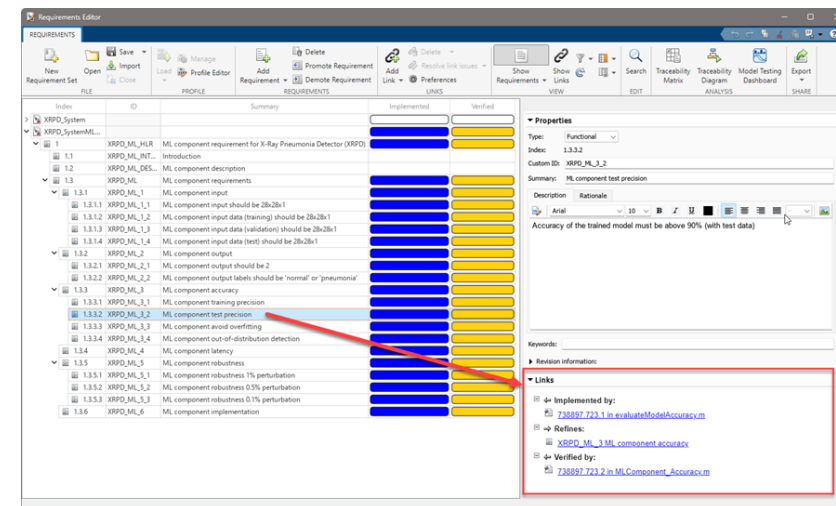


# Verifikace požadavků



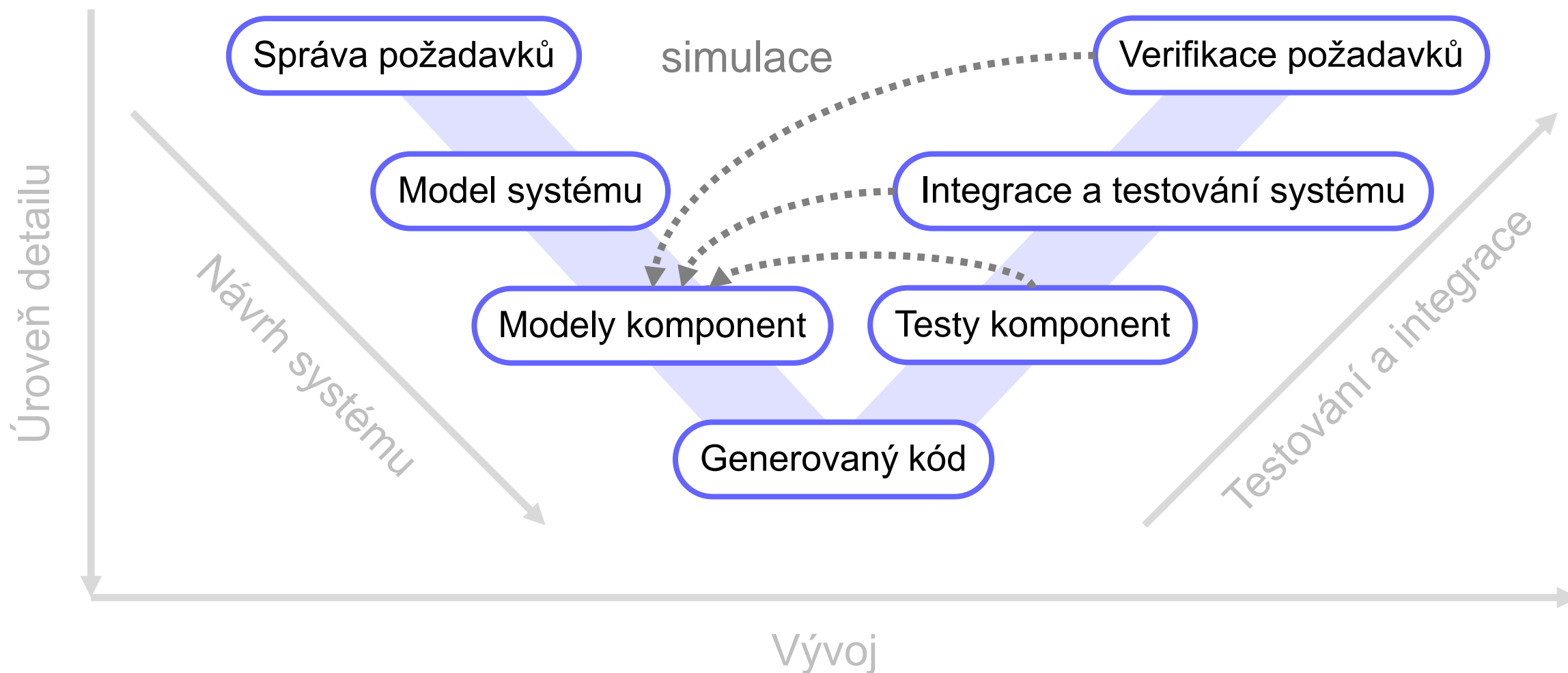
# Verifikace požadavků

- Závěrečná část V-cyklu
- Ověření, že byly veškeré požadavky
  - a) implementovány v podobě navrhované funkčnosti
  - b) splněny v podobě provedených testů
- Provázání požadavků s implementovanými funkcemi a testy
  - klíčové pro uzavření smyčky vývojového procesu
  - spuštěním všech definovaných testů ověříme, že požadavky byly adekvátně implementovány

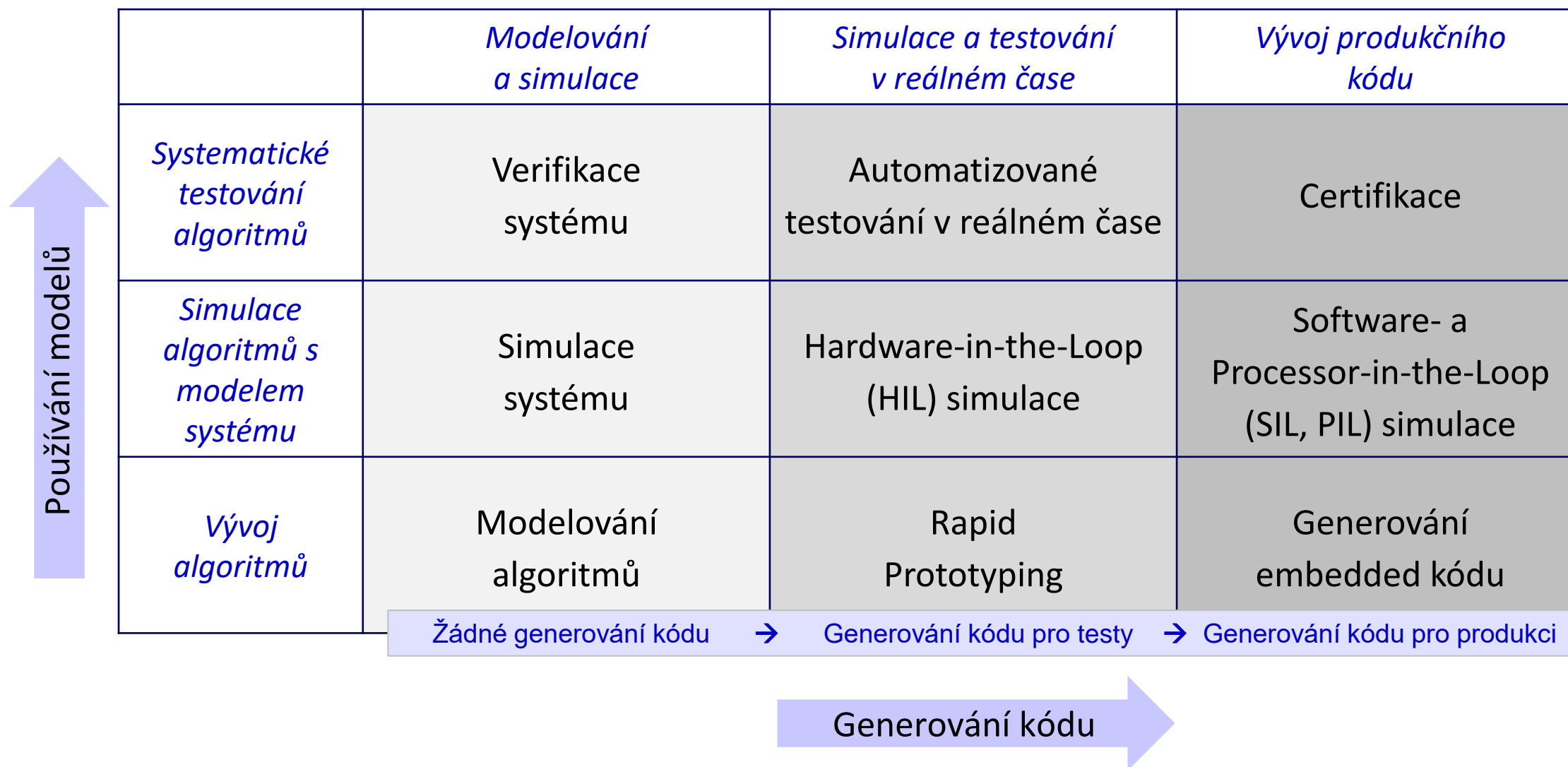


Index	ID	Summary	Implemented	Verified
> XRPD_System				
> XRPD_SystemMLC...				
1	XRPD_ML_HLR	ML component requirement for X-Ray Pneumonia Detector (XRPD)		
1.1	XRPD_ML_INT...	Introduction		
1.2	XRPD_ML_DES...	ML component description		
1.3	XRPD_ML	ML component requirements		
1.3.1	XRPD_ML_1	ML component input		
1.3.1.1	XRPD_ML_1_1	ML component input should be 28x28x1		
1.3.1.2	XRPD_ML_1_2	ML component input data (training) should be 28x28x1		

# Shrnutí: Vývojový V-cyklus pro (řídící) systémy s využitím MBD

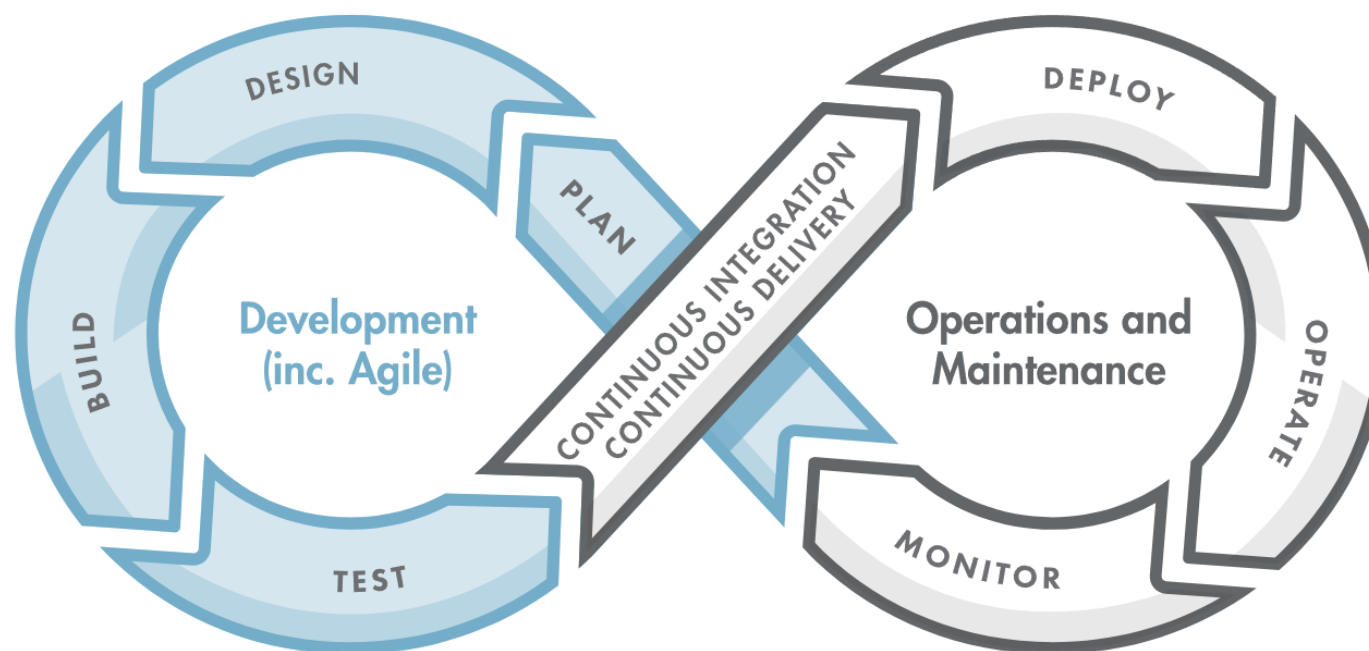


# Mapa nasazení metody MBD



# DevOps

- Zajišťujte průběžné aktualizace softwaru
- Iterace mezi vývojem a provozem systému
- Využívá simulace, automatizované testování a generování kódu



Děkuji za pozornost.

Otázky?

# Kontakty



- Pro technické dotazy využijte naši technickou podporu
  - email: [support@humusoft.cz](mailto:support@humusoft.cz)
  - tel.: +420 602 231 500
  - <https://www.humusoft.cz/matlab/support/>
- Všeobecné dotazy
  - email: [info@humusoft.cz](mailto:info@humusoft.cz)
  - tel.: +420 284 011 730
- Nebo můžete využít náš kontaktní formulář
  - <https://www.humusoft.cz/contact/#contact>

## Adresa:

Pobřežní 20  
186 00 Praha 8  
Česká republika

## Web:

[www.humusoft.cz](http://www.humusoft.cz)

