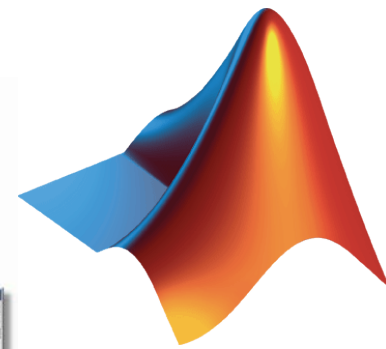
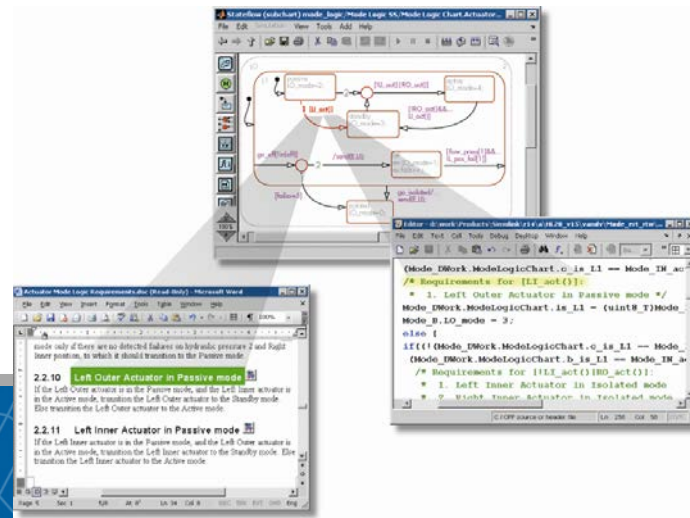


Improve Complexity Management with Model-Based Design in V-Modell



Polarion User-Conference 2013
Stuttgart, 01.10.2013

The MathWorks Team Today



Michael Hopfenzitz
Senior Account Manager



Christian Guß
Application Engineer

Agenda

- Introduction
- Complexity in Embedded Software Projects
- MathWorks User-Stories
- Model-Based Design Approach
- Traceability between Polarion Requirements and Simulink

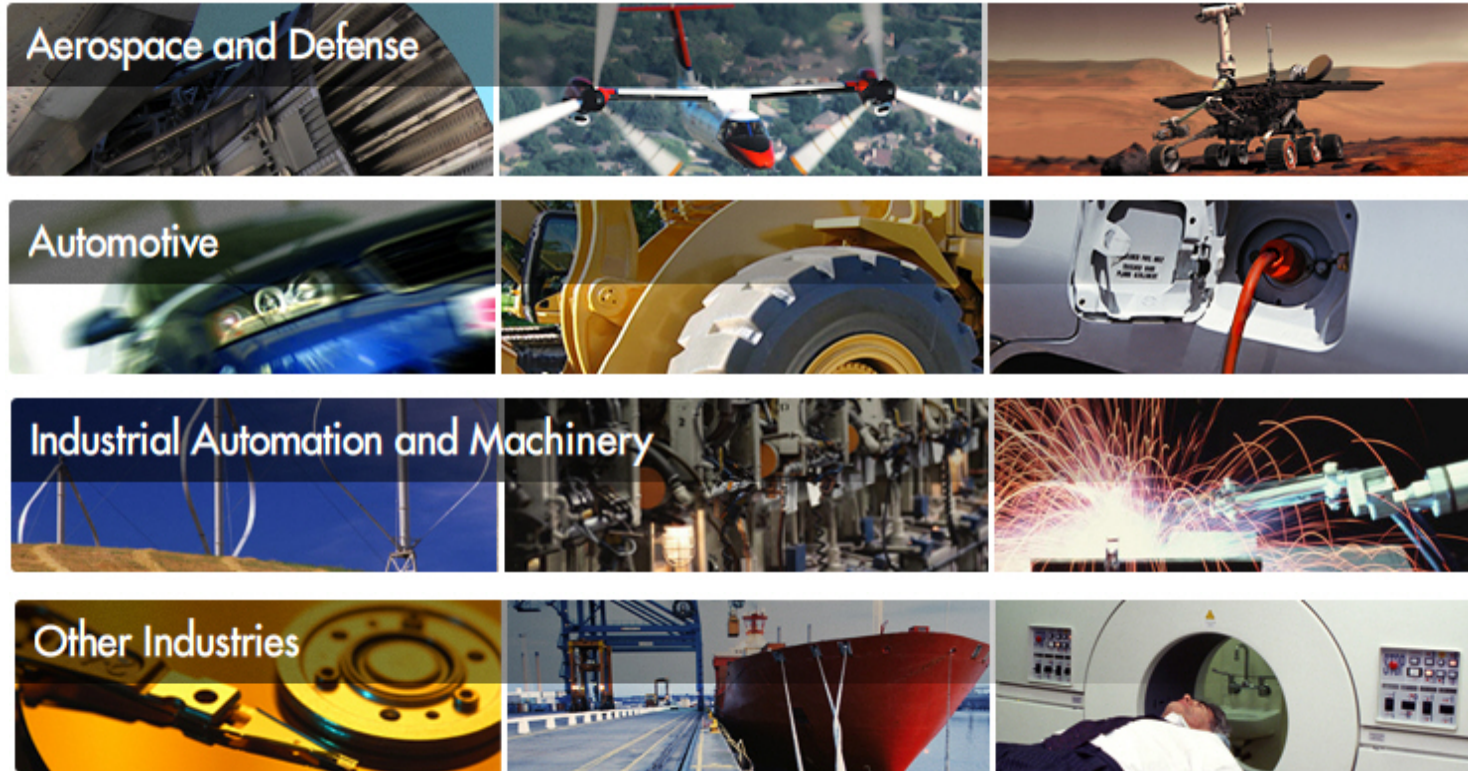
MathWorks at a Glance



Earth's topography on a Miller cylindrical projection, created with MATLAB and Mapping Toolbox

- **Headquarters:**
Natick, Massachusetts US
- **Other U.S. Locations:**
California, Michigan, Texas, Washington, DC
- **Europe:**
France, Germany, Italy, Netherlands, Spain, Sweden, Switzerland, United Kingdom
- **Asia-Pacific:**
Australia, China, India, Japan, Korea
- Worldwide training and consulting
- Distributors serving more than 20 countries

High-Integrity Applications



Software-based systems that are designed and maintained such that they have a high probability of carrying out their intended function

Weinmann Develops Life-Saving Transport Ventilator Using Model-Based Design



The MEDUMAT Transport ventilator.
Image © Weinmann Medical Technology.

Challenge

Develop embedded software for an advanced emergency and hospital transport ventilator

Solution

Use MATLAB and Simulink for Model-Based Design to model and simulate the controller, generate production code, and streamline compliance certification

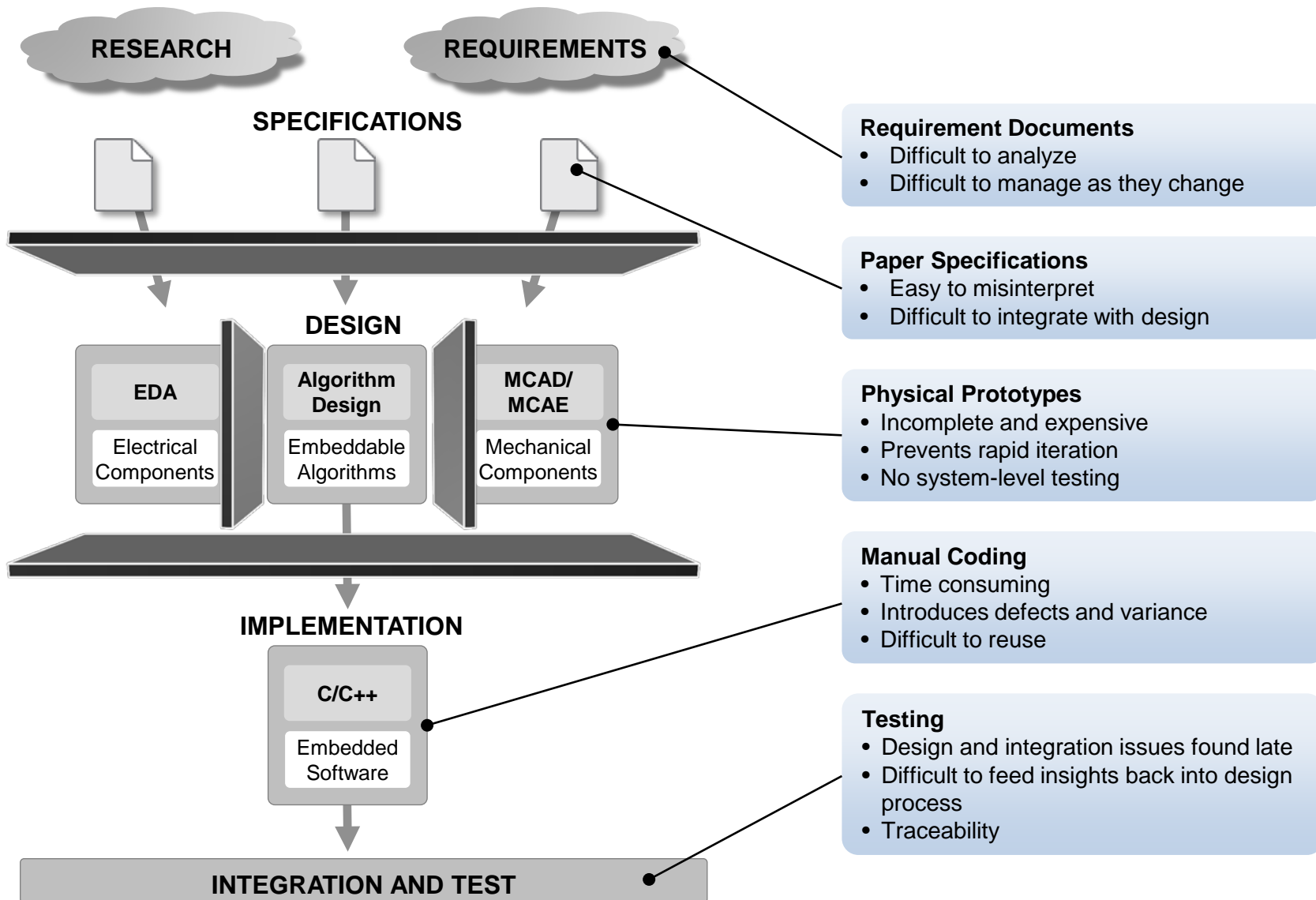
Results

- Code development and reviews accelerated by 50%
- Dozens of design alternatives explored
- 60% of core design reused

“Modeling, simulating, and implementing the ventilator’s embedded software with Simulink greatly simplified compliance certification. The model helped ensure a structured development process and provided thorough documentation and a visual representation of the system for the certification review.”

**Dr. Florian Dietz
Weinmann**

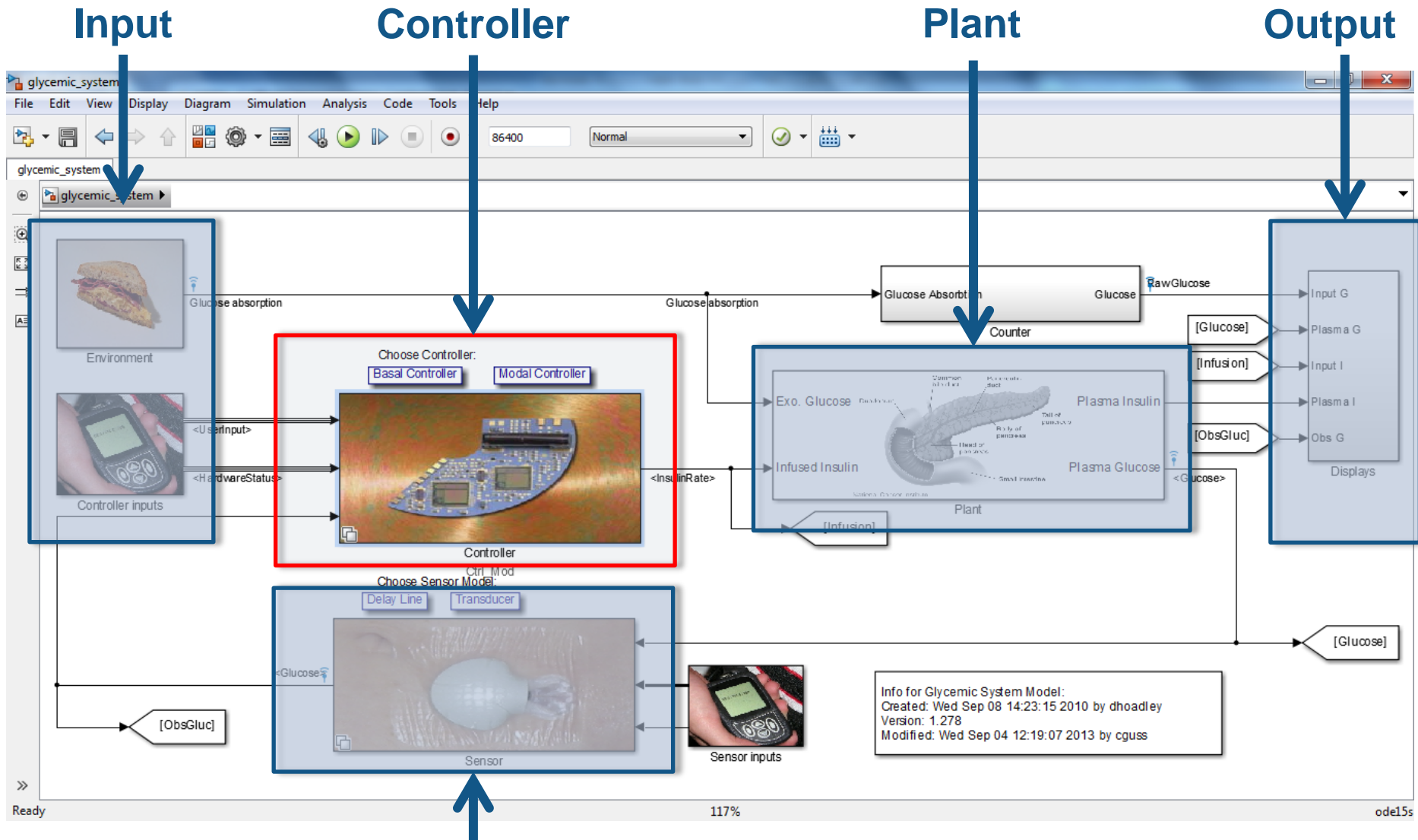
Complexity Challenges in Software Development



What is Model-Based Design?

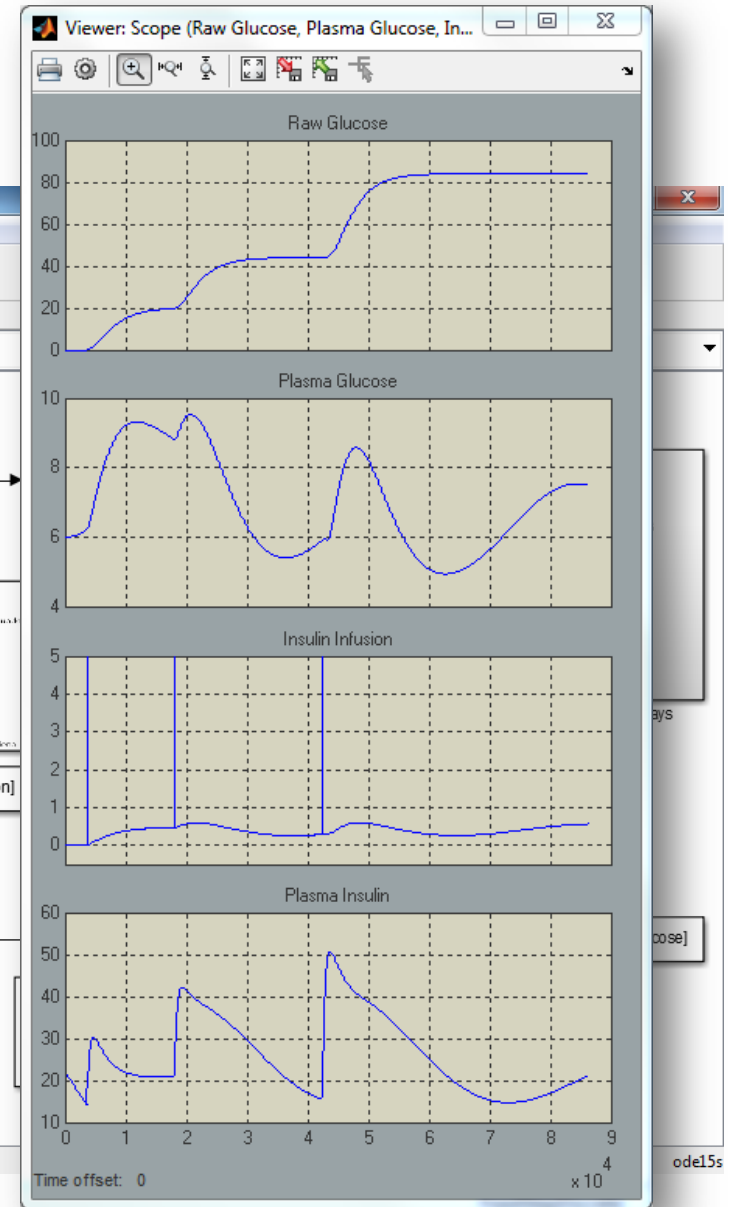
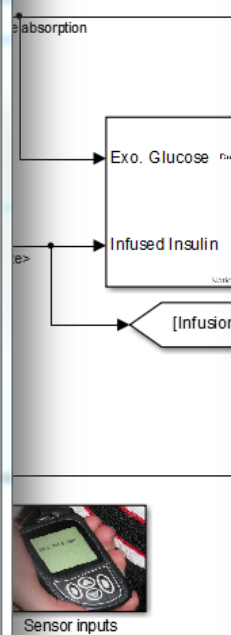
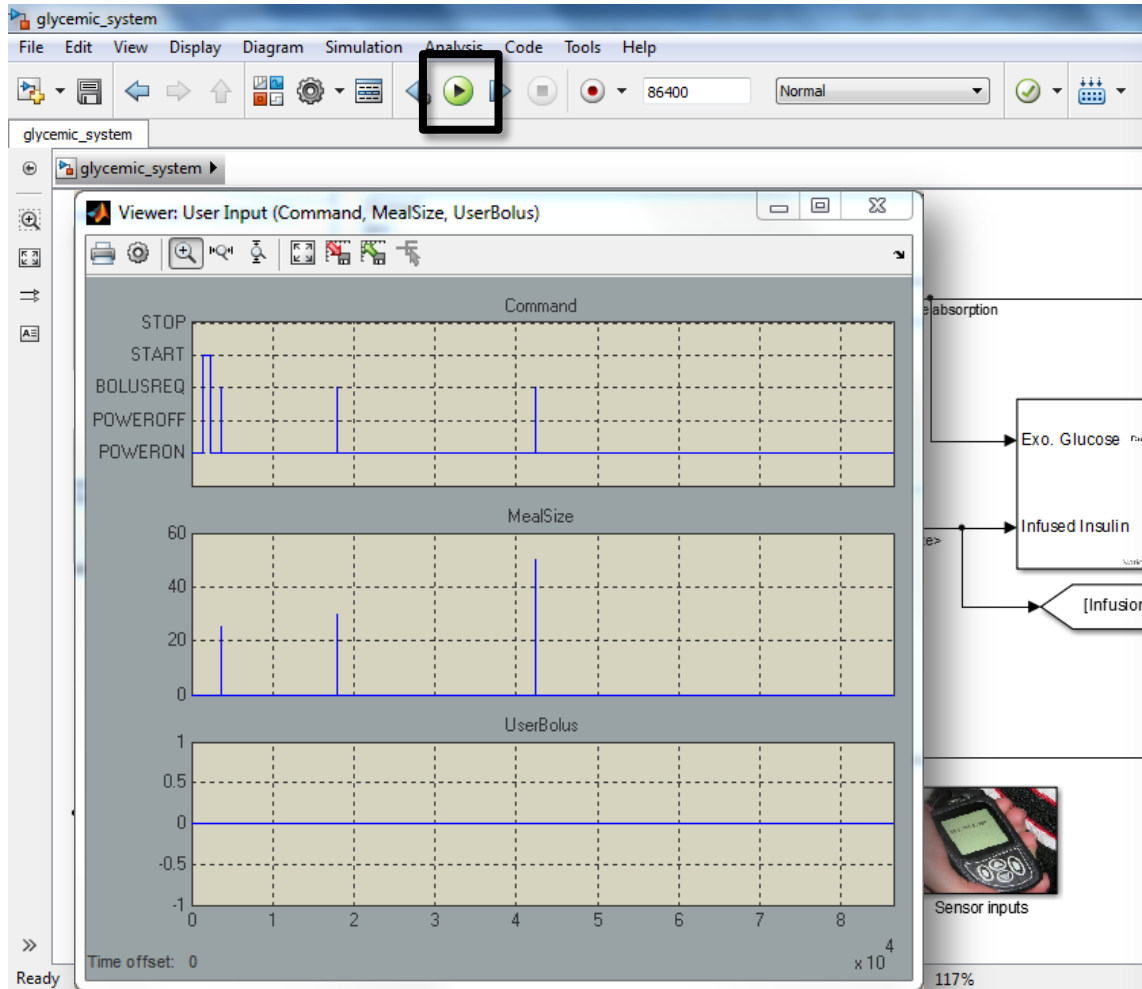
Model, Simulate, Verify the
control algorithm,
Auto-generate C code,
Deploy & Test on the
embedded hardware

Example: Glycemic Control System

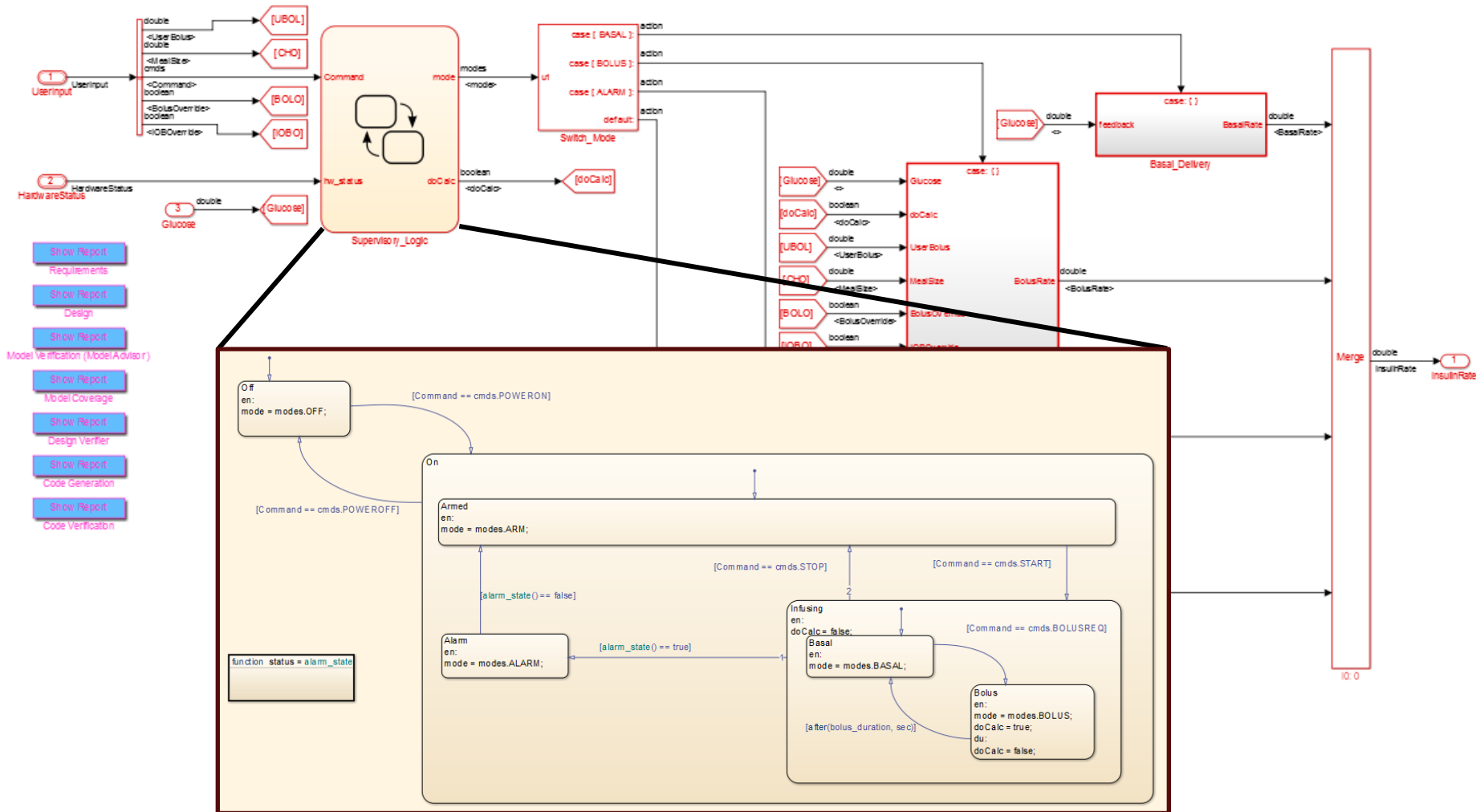


Sensor

Executable Specification

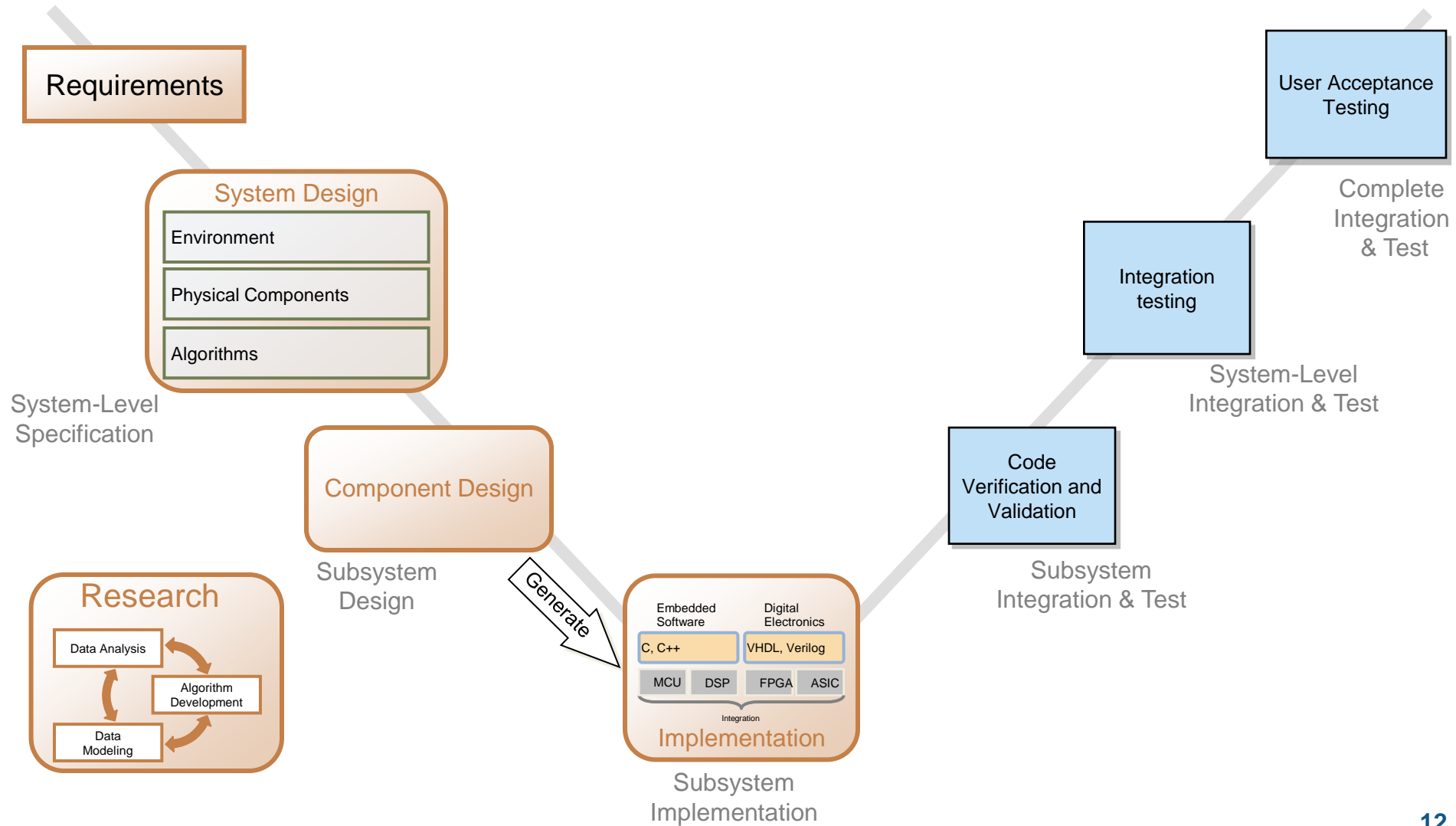


Component Design – Subsystems



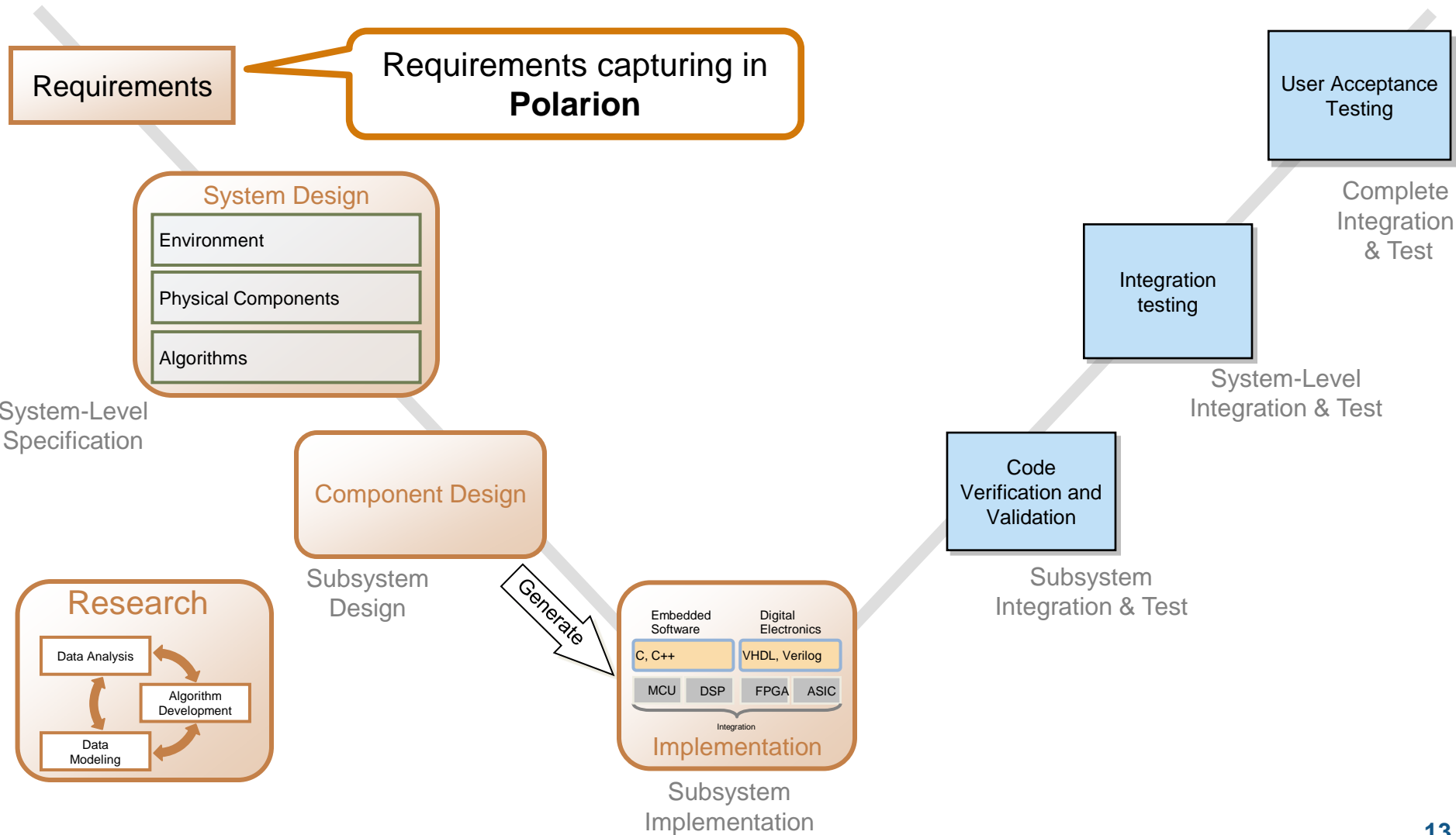
Model-Based Design

Development Process



Model-Based Design

Development Process



Polarion Connector for MATLAB Simulink

Polarion Connector for MATLAB® Simulink

[SHARE](#)

by Polarion



Polarion Connector for MATLAB® Simulink lets you link Simulink and State flow models with Polarion requirements and design items.

Added: Tue Jun 04 10:49:55 CEST 2013

Version: 1.0

Categories: Integration

[Download](#)

Certified Extension

See: www.polarion.com/connectors/matlab

Extension Requirements

- Polarion 2013
- MATLAB/Simulink 2013a
- Simulink Validation and Verification Toolbox

Highlight Requirements Inside

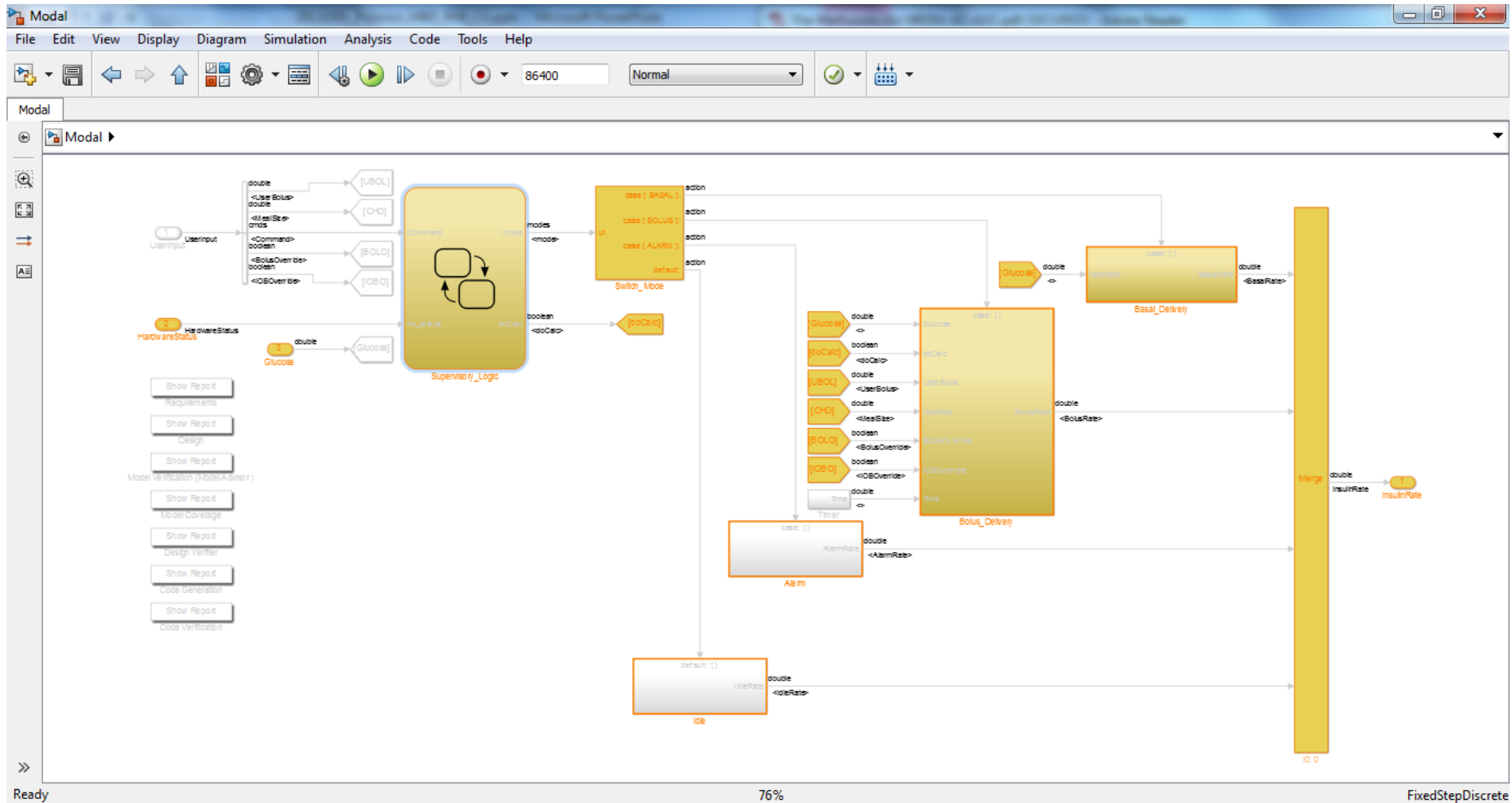
The screenshot displays the Simulink 'Modal' window. The 'Analysis' menu is open, with 'Requirements' selected. A sub-menu is visible, showing options like 'Load Links ...', 'Save Links', 'Save Links As ...', 'Copy to Model', 'Highlight Model', 'Check Consistency ...', 'Generate Report', 'Requirements at This Level', 'Requirements for Selected Object', and 'Settings ...'. The 'Highlight Model' option is currently selected and highlighted.

The background shows a Simulink block diagram with several blocks: 'Userinput', 'HardwareStatus', 'Supervisory_Logic', 'Glucose', 'Timer', 'Alarm', 'Bolus_Delivery', 'Merge', and 'InsulinRate'. The 'Glucose' block is highlighted with a red border. The 'InsulinRate' block is also highlighted with a red border. The 'Merge' block is highlighted with a red border. The 'InsulinRate' block is highlighted with a red border.

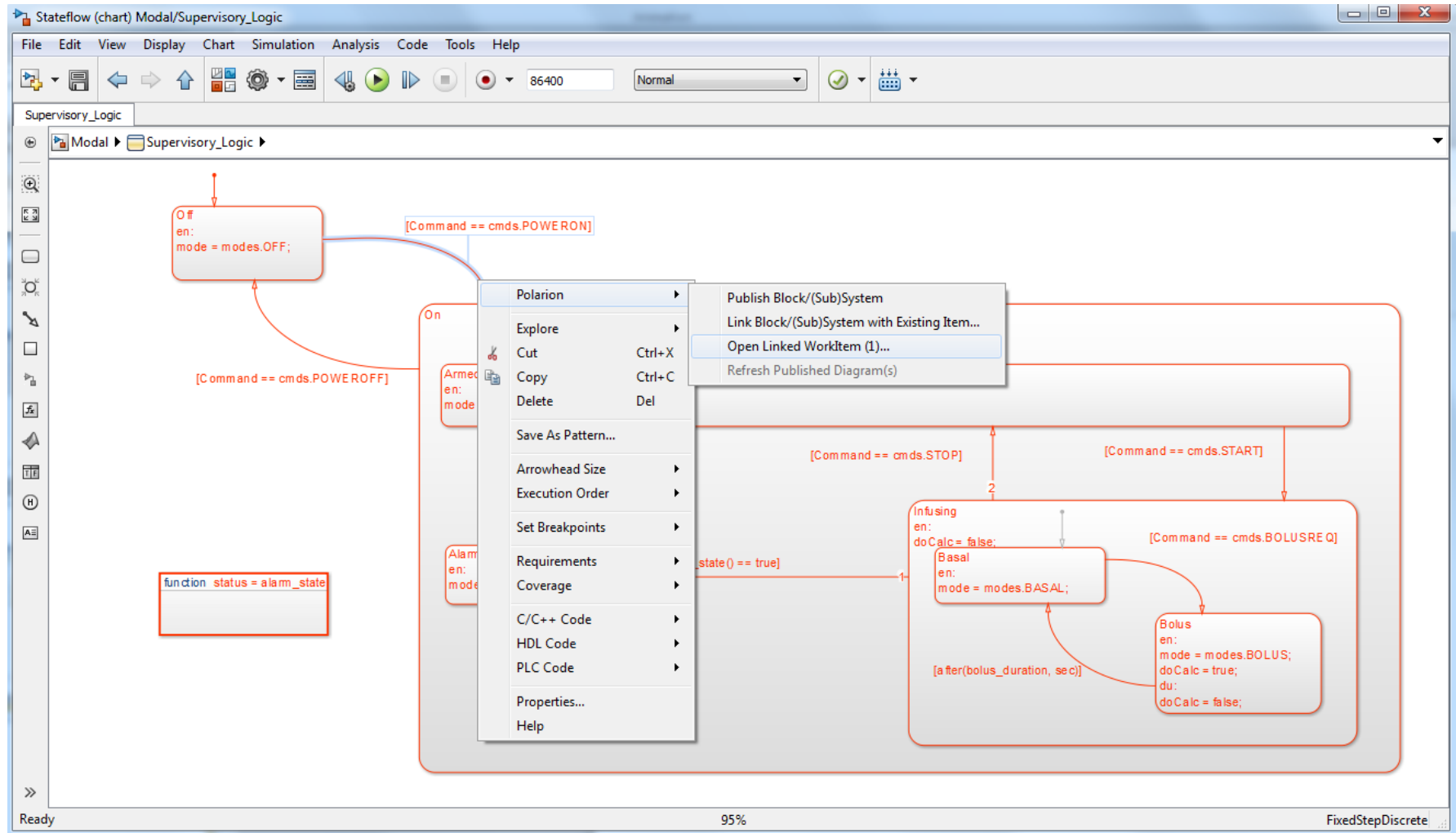
On the left side of the window, there is a list of 'Show Report' buttons for various analysis tools: Requirements, Design, Model Verification (Model Advisor), Model Coverage, Design Verifier, Code Generation, and Code Verification.

The status bar at the bottom indicates 'Ready', '76%', and 'FixedStepDiscrete'.

Highlight Requirements Inside



Traceability from Simulink to Polaron



Traceability from Simulink to Polarion

Stateflow (chart) Modal/Supervisory_Logic

File Edit View Display Chart Simulation Analysis Code Tools Help

Supervisory_Logic

Modal ▸ Supervisory_Logic ▸

Off
en:
mode = modes.OFF;

[Command == cmds.POWERON]

[Command == cmds.POWEROFF]

function status = alam_state

On
en:
mode

Alarm
en:
mod

command == cmds.START]

[Command == cmds.BOLUSREQ]

Bolus
en:
mode = modes.BOLUS;
doCalc = true;
du:
doCalc = false;

Linked Work Items

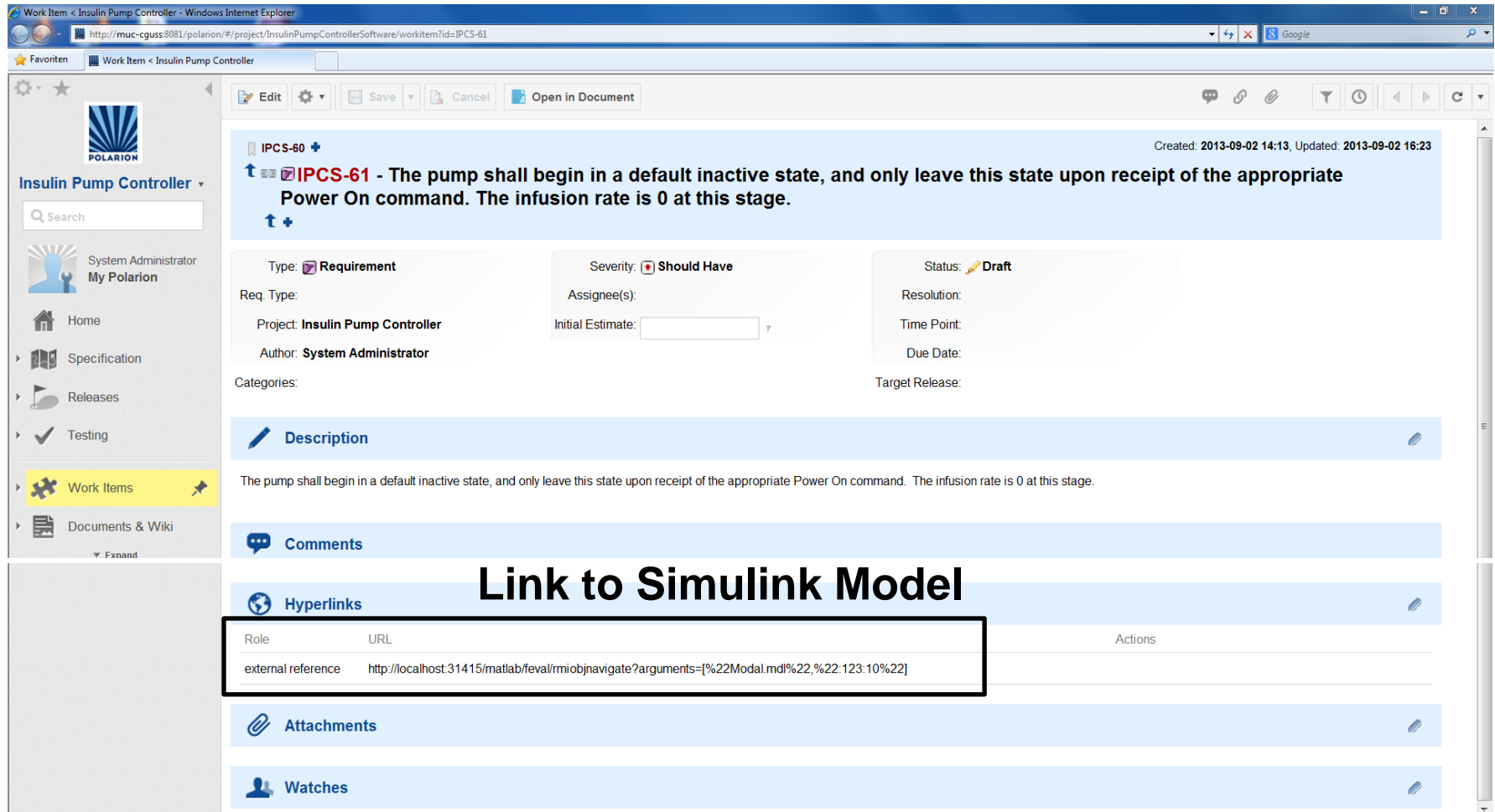
ID	Title
IPCS-61	The pump shall begin in a default inactive state, and only leav...

Click on an item to open it in browser

OK

Ready 95% FixedStepDiscrete

Traceability from Polarion to Simulink



The screenshot shows a web browser window displaying a Polarion work item. The browser address bar shows the URL: `http://muc-cguss:8081/polarion/#/project/InsulinPumpControllerSoftware/workitem?id=IPCS-61`. The work item title is **IPCS-61 - The pump shall begin in a default inactive state, and only leave this state upon receipt of the appropriate Power On command. The infusion rate is 0 at this stage.** The item is of type **Requirement**, severity **Should Have**, and status **Draft**. The author is **System Administrator**. The description field contains the text: "The pump shall begin in a default inactive state, and only leave this state upon receipt of the appropriate Power On command. The infusion rate is 0 at this stage." Below the description is a **Hyperlinks** section with a table:

Role	URL	Actions
external reference	http://localhost:31415/matlab/feval/rmiobjnavigate?arguments=[%22Modal.mdl%22,%22:123:10%22]	

The table row is highlighted with a black border. Other sections visible include **Description**, **Comments**, **Attachments**, and **Watches**. The left sidebar shows the application navigation menu with 'Work Items' selected.

Model-Based Design

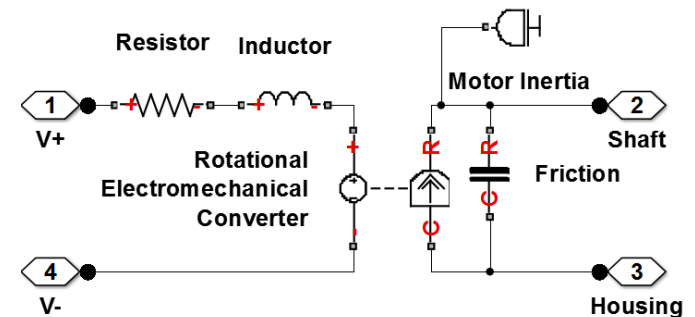
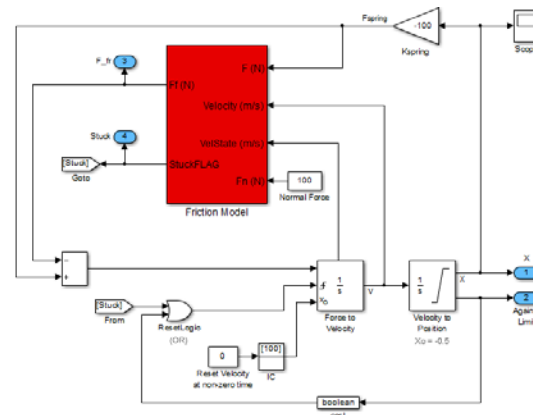
Multi-Domain Modeling and Algorithm Development

Requirements

System Design

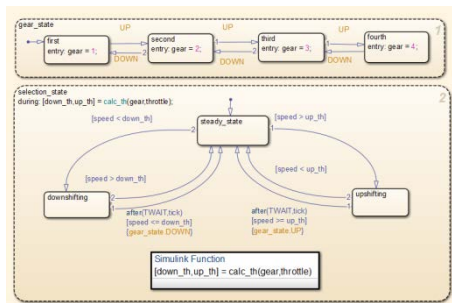
- Environment
- Physical Components
- Algorithms

Methods for modeling systems in different domains



Data Flow (Block diagram)

Physical Modeling (Schematic)



Modeling of Event-Driven Systems (State - Machines)

Programming Language (Textual)

```
function [apphls, weights] = gaincontrol(rewip, train)
% 3-tap adaptive equalizer using RLS algorithm

% Signal settings
lambda = 0.99; % forgetting factor for RLS

% Initialization
Delta = 0.1463; % var. cov. parameter
weights = zeros(1,3);

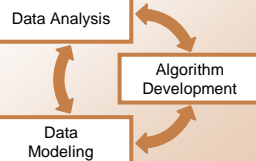
apphls = zeros(length(rewip),3);
for n = 1:length(rewip)
    u = xsig(n); % received sample
    y = conv(weights, u); % output
    if n>length(rewip)
        % training code
        d = train(n);
    else
        % detection-oriented code
        d = detect(preval(y)) + 0.5*detect(logp(y));
    end
    % single-tap RLS
    Delta = 1/(lambda/Delta + u'*u*(n));
    d = Delta * u;
    e = d - y; % aperiodic estimation error
    weights = weights + d*conj(u)'; % update weights
    apphls(n) = y;
end
```

System-Level Specification

Control

Subsystem Design

Research



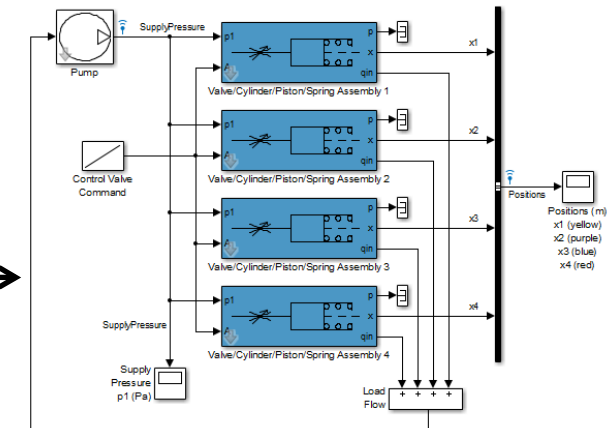
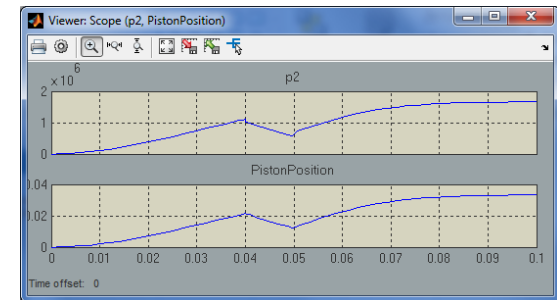
Model-Based Design

Early Concept Verification

Requirements

User Acceptance Testing

- Executable specifications
- Predict dynamic system behaviour by simulation
 - System & environment models
 - Less physical prototypes
- Use of simulation results for system design
 - Fast What-/If studies
 - Short iteration cycles



Simple Model

Detailed Model



Idea



System-Level Specification

Research

Data Analysis

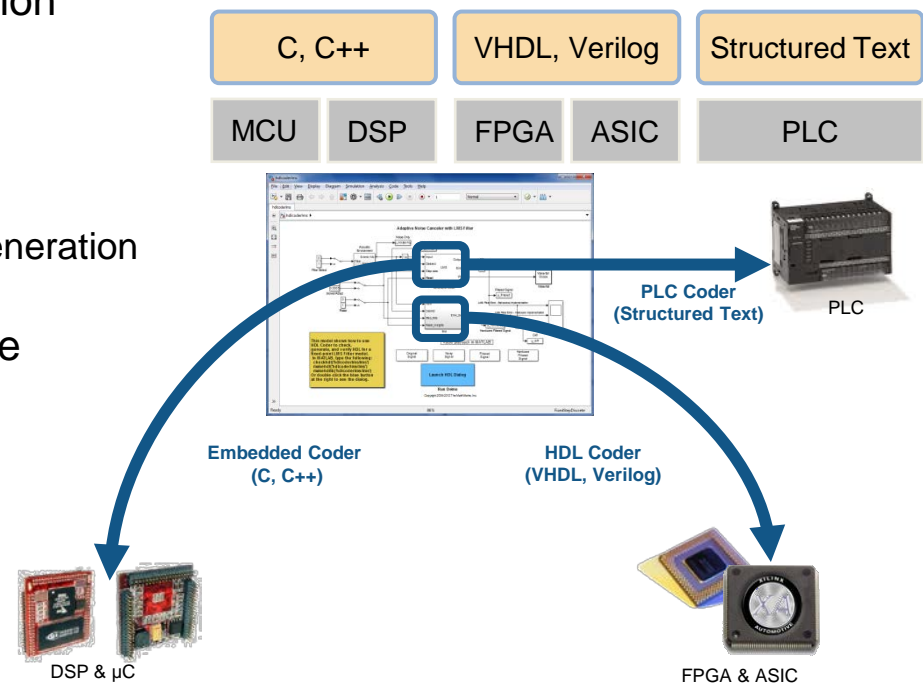
App Development

Data Modeling

Model-Based Design

Automatic Code Generation

- C/C++, VHDL and PLC-Code Generation from **one model**
- Support for Fixed Point Data Format
 - Automatic scaling
 - Supported in Simulation and Code-Generation
- Easy integration of legacy C/C++-Code
- System development independent of the target

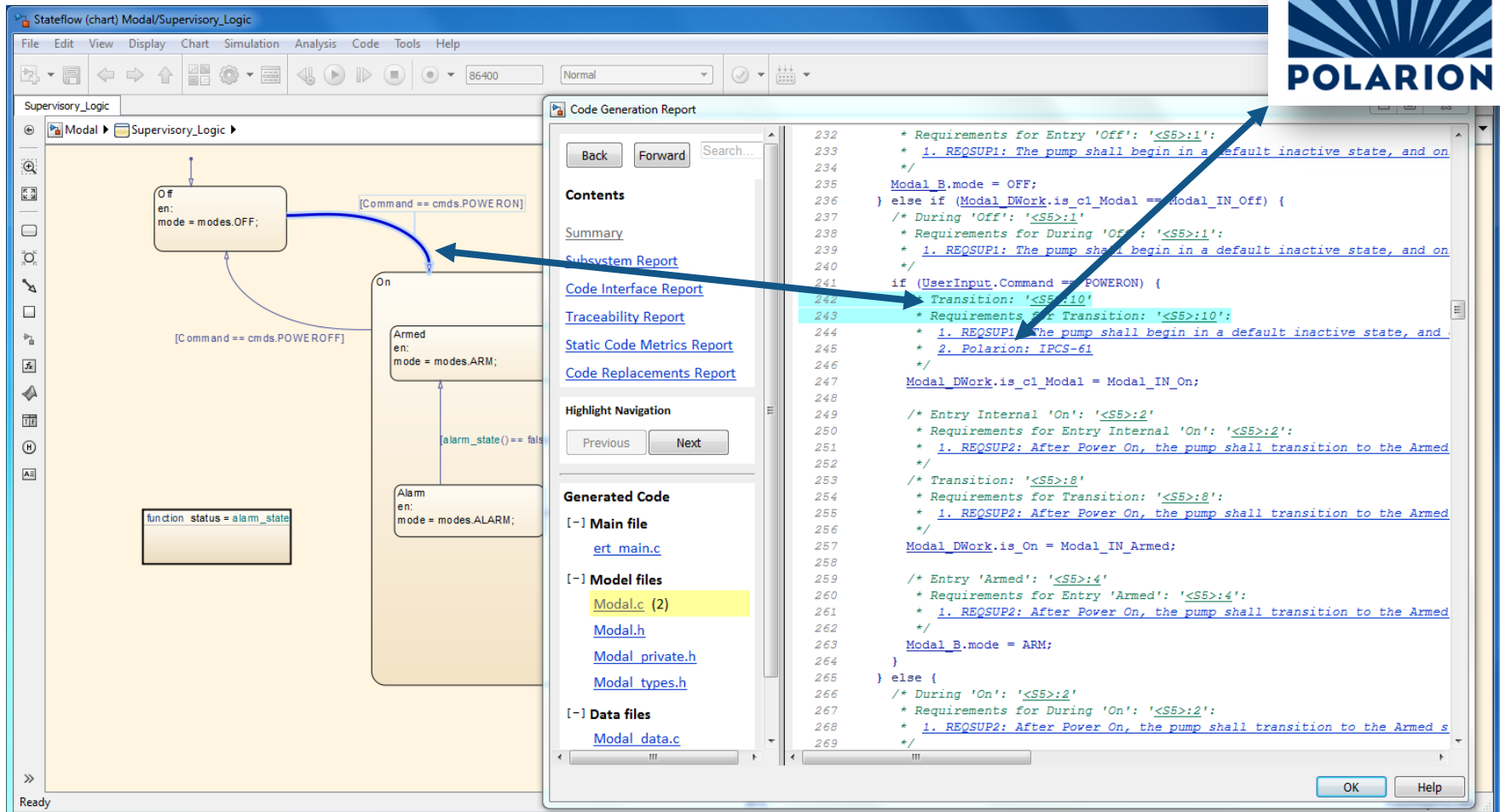


Integration
Implementation
 Subsystem Implementation



Traceability: Requirement \leftrightarrow Model \leftrightarrow Code

HTML Code Generation Report

The screenshot displays the Stateflow environment for a 'Supervisory_Logic' model. On the left, a state machine diagram shows three states: 'Off', 'Armed', and 'Alarm'. Transitions are labeled with conditions like '[Command == cmds.POWERON]' and '[alarm_state() == false]'. A function block 'status = alarm_state' is also visible.

On the right, the 'Code Generation Report' is open. The 'Contents' pane lists various reports, with 'Traceability Report' selected. The 'Generated Code' pane shows the code for the transition from 'Off' to 'Armed'. The code includes requirements for the transition, such as '1. REQSUP1: The pump shall begin in a default inactive state, and on' and '1. REQSUP2: After Power On, the pump shall transition to the Armed'. These requirements are highlighted in blue, and arrows point from them back to the corresponding transition in the state machine diagram.

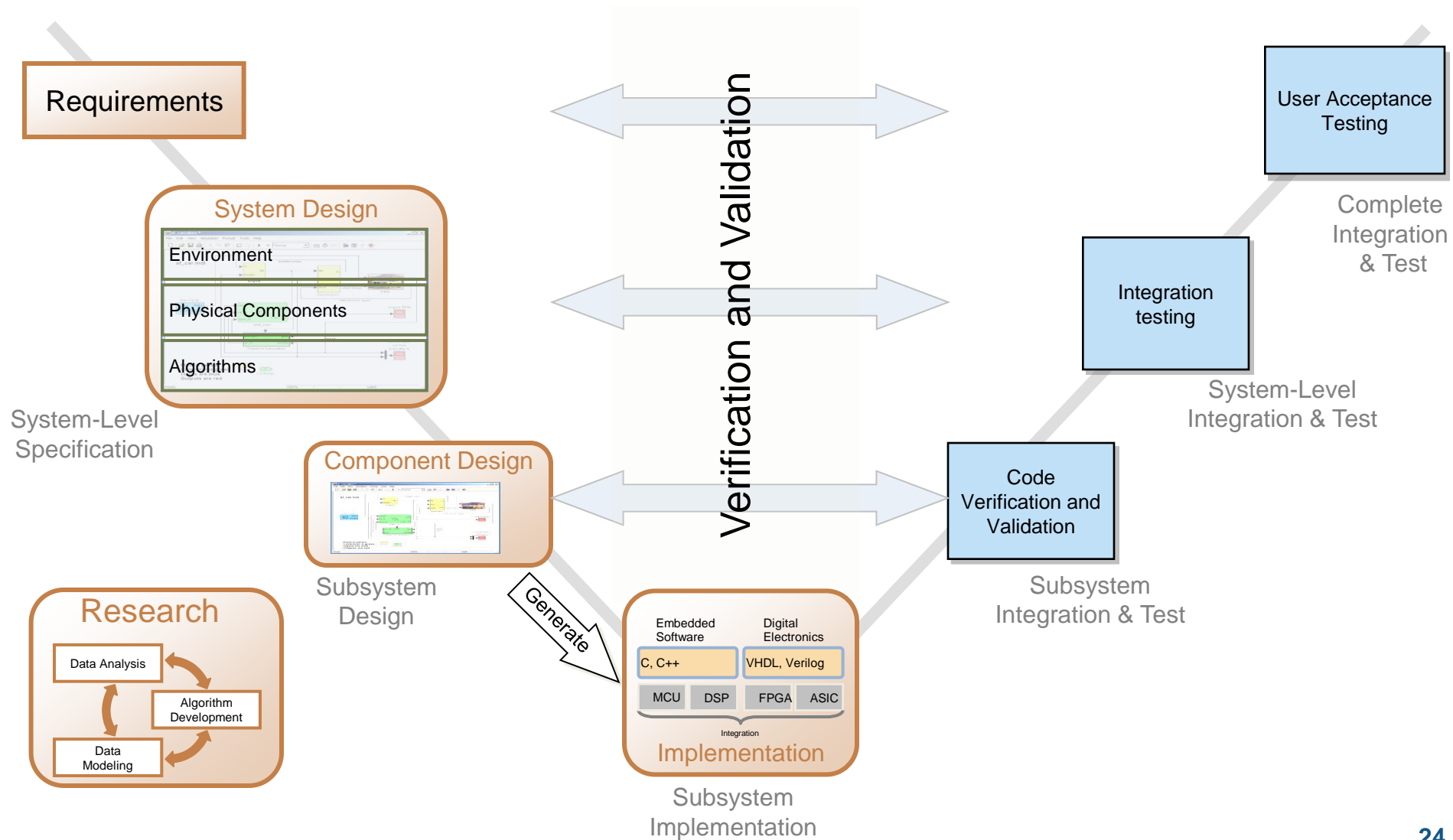
```

232  * Requirements for Entry 'Off': '<SS>:1':
233  * 1. REQSUP1: The pump shall begin in a default inactive state, and on
234  */
235  Modal_B.mode = OFF;
236  } else if (Modal_DWork.is_c1_Modal == Modal_IN_Off) {
237  /* During 'Off': '<SS>:1'
238  * Requirements for During 'Off': '<SS>:1':
239  * 1. REQSUP1: The pump shall begin in a default inactive state, and on
240  */
241  if (UserInput.Command == POWERON) {
242  Transition: '<SS>:10'
243  * Requirements for Transition: '<SS>:10':
244  * 1. REQSUP1: The pump shall begin in a default inactive state, and
245  * 2. Polarion: IPCS-61
246  */
247  Modal_DWork.is_c1_Modal = Modal_IN_On;
248
249  /* Entry Internal 'On': '<SS>:2'
250  * Requirements for Entry Internal 'On': '<SS>:2':
251  * 1. REQSUP2: After Power On, the pump shall transition to the Armed
252  */
253  /* Transition: '<SS>:8'
254  * Requirements for Transition: '<SS>:8':
255  * 1. REQSUP2: After Power On, the pump shall transition to the Armed
256  */
257  Modal_DWork.is_On = Modal_IN_Armed;
258
259  /* Entry 'Armed': '<SS>:4'
260  * Requirements for Entry 'Armed': '<SS>:4':
261  * 1. REQSUP2: After Power On, the pump shall transition to the Armed
262  */
263  Modal_B.mode = ARM;
264  }
265  } else {
266  /* During 'On': '<SS>:2'
267  * Requirements for During 'On': '<SS>:2':
268  * 1. REQSUP2: After Power On, the pump shall transition to the Armed s
269  */

```

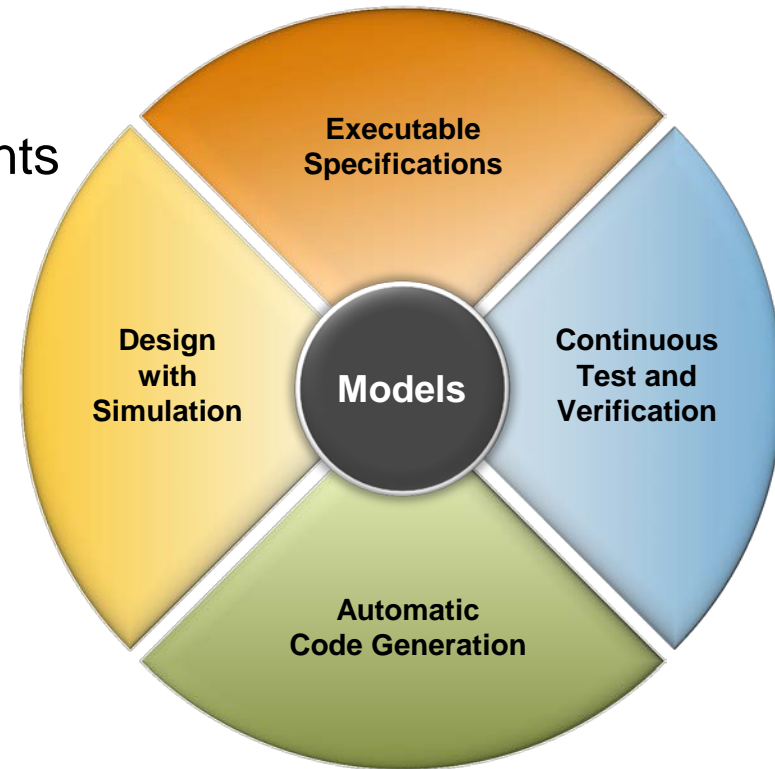
Model-Based Design

Continuous Verification and Validation



Benefits of Model-Based Design

- Models:
 - Core of the Development Process**
 - Unambiguous Description of Requirements (Executable Specification)
 - Fast Evaluation of Design Variants
 - Frontloading - Early Test and Verification
 - Automatic Code Generation
- ⇒ **Better Cooperation, Communication and Collaboration**
- ⇒ **Higher Product Quality**



Support and Community

 MathWorks® | *Application Engineering*

 MathWorks® | *Training Services*

 **MATLAB**® **CENTRAL**

 MathWorks® | *Consulting Services*

 MathWorks® | *Connections Program*

 MathWorks® | *Technical Support*