CEC Continuing Education Center

DesignNews

Real-Time System Software Architecture Design

DAY 2 : Design Philosophies and Principles

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

THE SPEAKER

Jacob Beningo

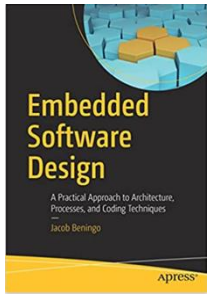
Jacob@beningo.com

jacobbeningo

Beningo Embedded Group – CEO / Founder

Focus: Software Architecture, Processes, and Dev Skills

At Beningo Embedded Group, we believe everyone deserves the skills to confidently advance their careers, meet deadlines, and deliver quality embedded systems. We provide modern strategies, insights, and hands-on training to equip developers and teams with the tools they need to succeed.



EMBEDDED SOFTWARE ACADEMY BY BENINGO

Visit **www.beningo.com** to learn more

This week's topics:

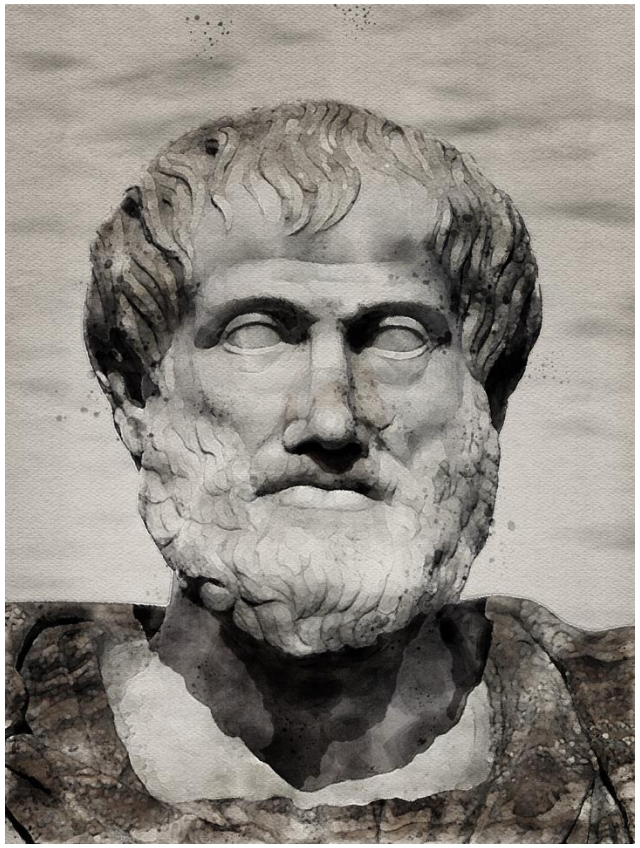
- What is Software Architecture?
- Design Philosophies and Principles
- Modeling with UML and the 4C Model
- Data-Centric Architecture Design
- Beyond UML – Data, Isolation, Security

01 Design Philosophy and Principles

Defining your Design Philosophy

A design philosophy is a practical set of ideas about designing a system that encompasses a set of guiding principles and practices that prioritize the unique constraints and requirements of embedded systems.

- Critical challenges should be addressed such as:
  - Cost
  - Quality
  - Time to market
  - Scalability
  - Portability
  - Etc



Philosophy versus Principles

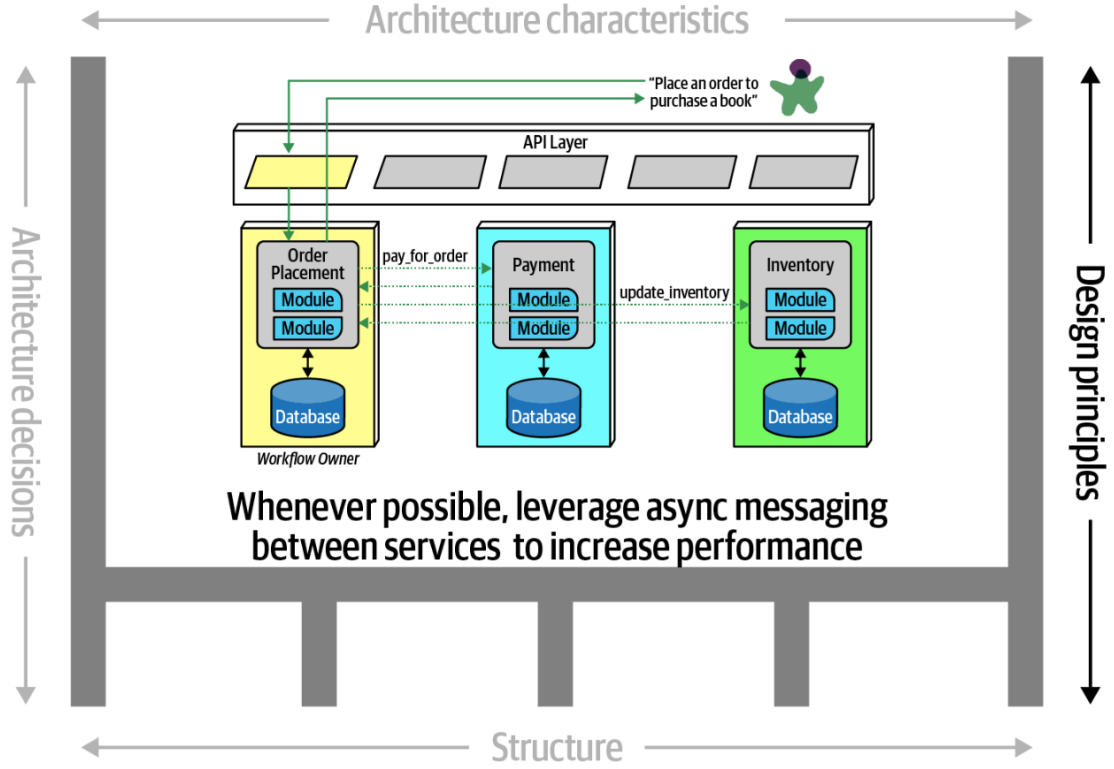
The difference lies in their scope and application:

1. **Philosophy** – Reflects a high-level perspective or approach to a field, discipline, or life in general.
    1. Example: "Embedded systems should be hardware-agnostic to maximize portability."
2. **Principle** – A specific guiding rule or fundamental truth derived from a philosophy. Principles are more concrete, actionable, and serve as practical guidelines for making decisions.
    1. Example (from the first philosophy): "Use HALs or abstraction layers to decouple software from hardware dependencies."

Aspect	Philosophy	Principle
Scope	Broad, conceptual	Narrower, actionable
Nature	Abstract belief system	Concrete rule or guideline
Purpose	Provides a worldview or approach	Directs specific actions
Example	“Software should be simple and reliable”	“Minimize dependencies to reduce complexity”

Design Principles Review

- Guiding rules that shape software architecture decisions.
- Ensure scalability, maintainability, and flexibility.
- Help manage complexity and improve system reliability.
- Balance performance, security, and adaptability.
- Provide a structured approach to building robust software.



Source: Fundamentals of Software Architecture

Audience POLL Question

What do you think about defining your software philosophy and principles?

a) Waste of time
b) Somewhat useful to identify our guiding principles
c) Critical to ensure every developer is on the same page
d) Other

01 Modern Design Philosophies

What are your design philosophies?

- Fight the biggest fire first

Not a good design philosophy!

7 Philosophies For Embedded Software Design

- Define the principles that are most important to you.

Philosophy #1 – Data Dictates Design

Philosophy #2 – There is No Hardware (only data)

Philosophy #3 – KISS the Software

Philosophy #4 – Practical, Not Perfect

Philosophy #5 – Future-proofing ensures flexibility

Philosophy #6 – Quality is a mindset, not an afterthought

Philosophy #7 – Security isn't optional–it's fundamental

Action Item:

Identify and write down your top 7 philosophies and principles.

Put them in a place you can readily see them!

Solid Principles

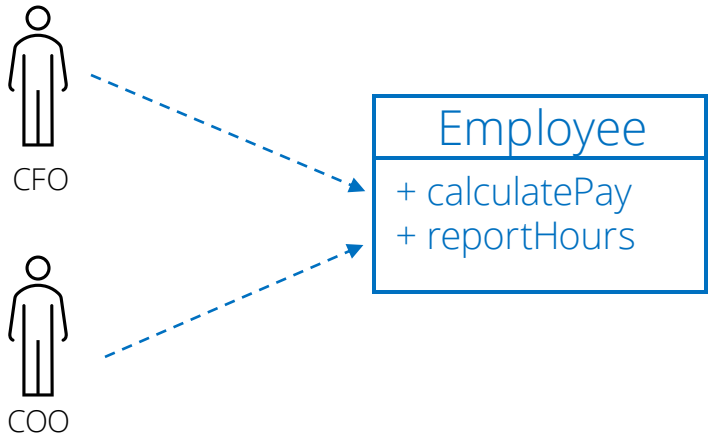
SOLID principles tell developers how to arrange their functions and data structures into classes and how they should be interconnected.

The goal is to create software groupings that:

- Tolerate change
- Are easy to understand
- Create components that can be used in many systems
- Easily tested
- Minimize code changes

SRP is violated because the employee class is responsible for two different users!

- Accounting department
- Human Resources



Audience POLL Question

How familiar are you with SOLID Principles?

a) Never heard of them.
b) Heard of them, but don't use them.
c) I actively use them to design systems
d) I live and breath SOLID

SOLID Princinples

03

Overview of SOLID

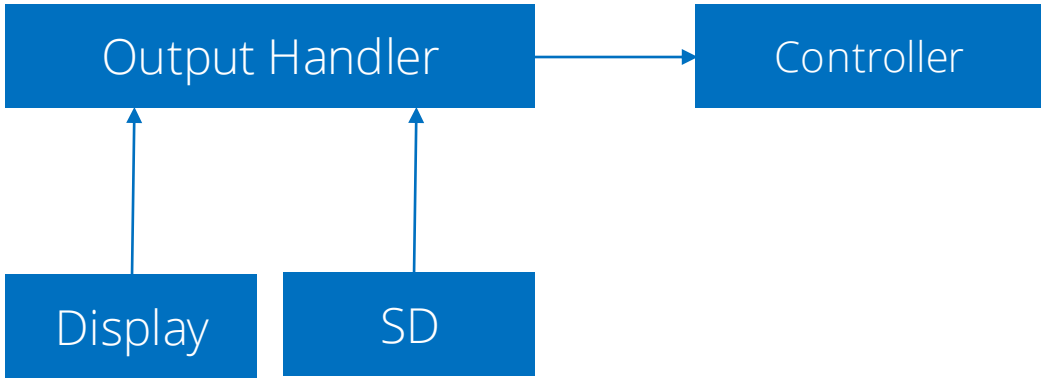
SRP: The Single Responsibility Principle - Each software module has one, and only one, reason to change.

OCP: The Open-Closed Principle - For software systems to be easily changed, they must add new code rather than change existing code.

LSP: The Liskov Substitution Principle - To build software systems from interchangeable parts, those parts must adhere to a contract allowing substitution.

ISP: The Interface Segregation Principle - Advises software designers to avoid depending on things they don't use.

DIP: The Dependency Inversion Principle - High-level application code should not be dependent on low-level implementation details.



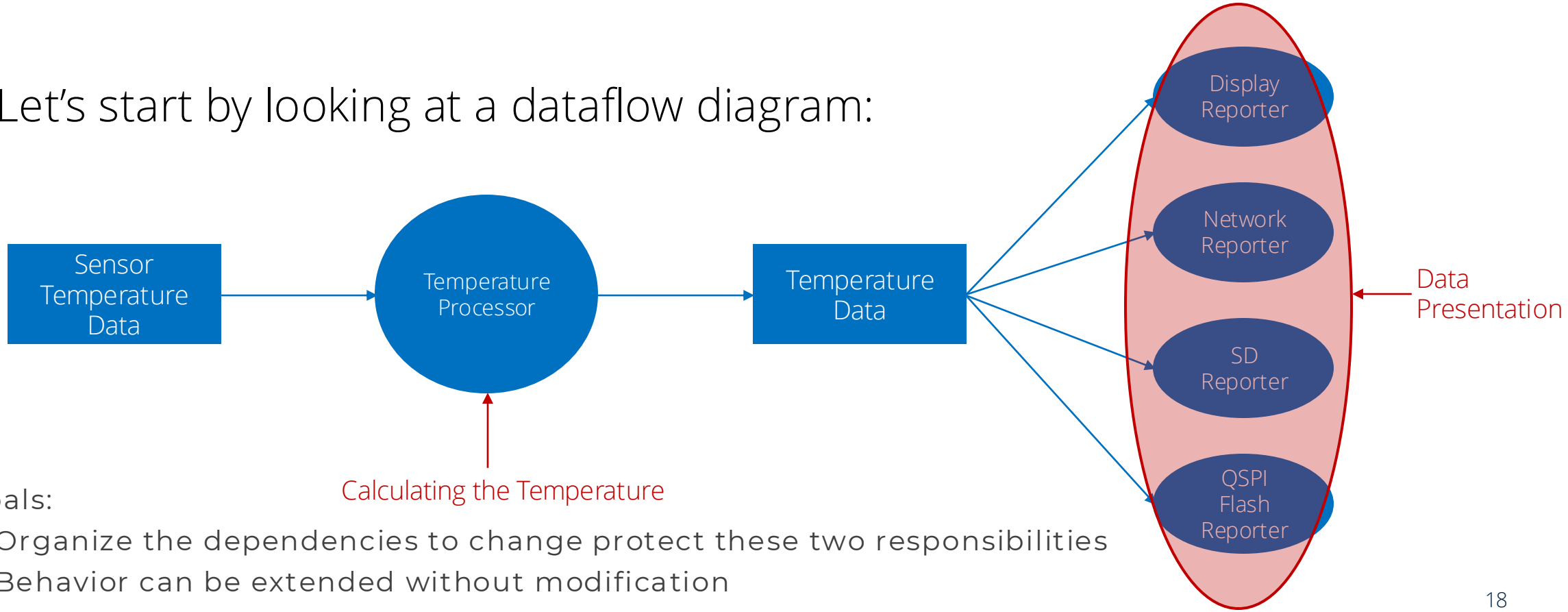
Why Should You Care?

SOLID principles guide developers to create object-oriented designs that:

- Promote cohesion
- Facilitate testing
- Enhance flexibility
- Encourage modular design
- Improves interchangeability
- Enhances reliability
- Reduces unnecessary dependence
- Improves code understandability
- Decouples high-level and low-level code

OCP: The Open-Closed Principle Example

- Let's start by looking at a dataflow diagram:

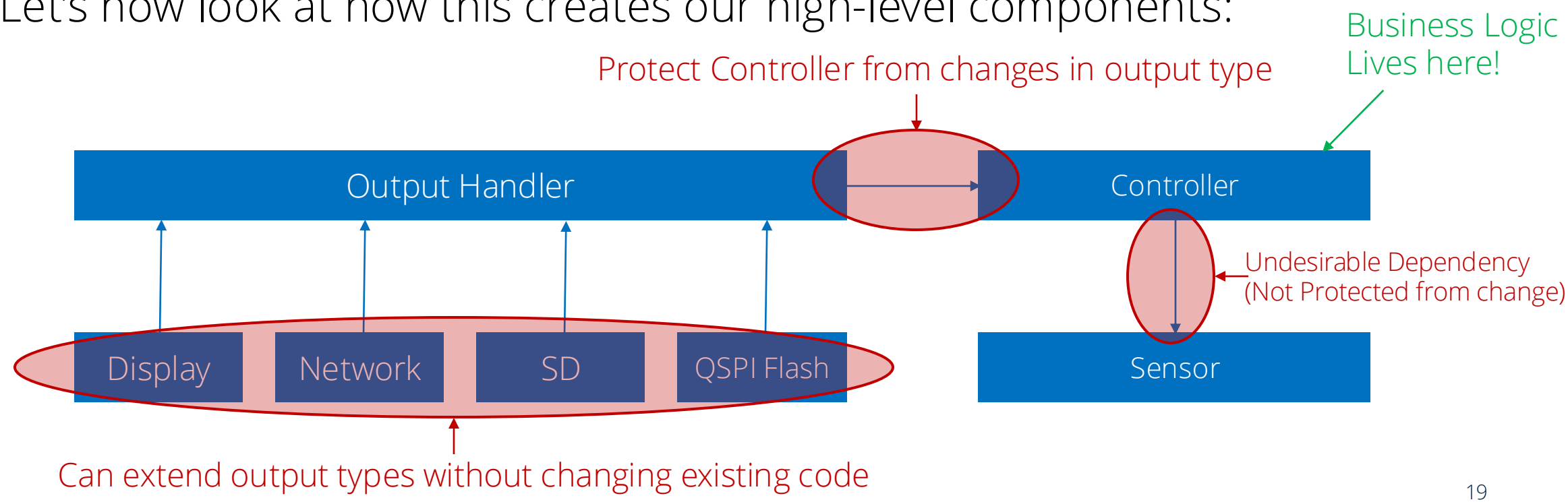


Goals:

- Organize the dependencies to change protect these two responsibilities
- Behavior can be extended without modification

OCP: The Open-Closed Principle Example

- Let's now look at how this creates our high-level components:



Audience POLL Question

Which of the following best describes the Dependency Inversion Principle (DIP) in software design?

a) High-level modules should depend on low-level modules, which in turn depend on hardware.
b) High-level modules should not depend on low-level modules; both should depend on abstractions.
c) Low-level modules should depend on concrete implementations provided by high-level modules.
d) Abstractions should depend on details, since they define the system's behavior.

Next Steps

04

Going Further

Download the extra resources:

- https://beningo.short.gy/25DNCEC-10-Architecture-Resources

Get Hands-On:

- Analyze an existing projects software architecture
- Draw a new software architecture
- Invest a widget to practice your architecture skills
- Join an Embedded Software Architecture Kata

Downloadable Resources:

- Characteristics Worksheet
- Modern Principles
- Architecture Book List
- ADR Template

Save $100:
https://beningo.short.gy/DNCEC2510

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

Consulting | Coaching | Training

www.beningo.com

Next Steps

- ✓ What is Software Architecture?
- ✓ Design Philosophies and Principles
- Modeling with UML and the 4C Model
- Data-Centric Architecture Design
- Beyond UML – Data, Isolation, and Security

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets