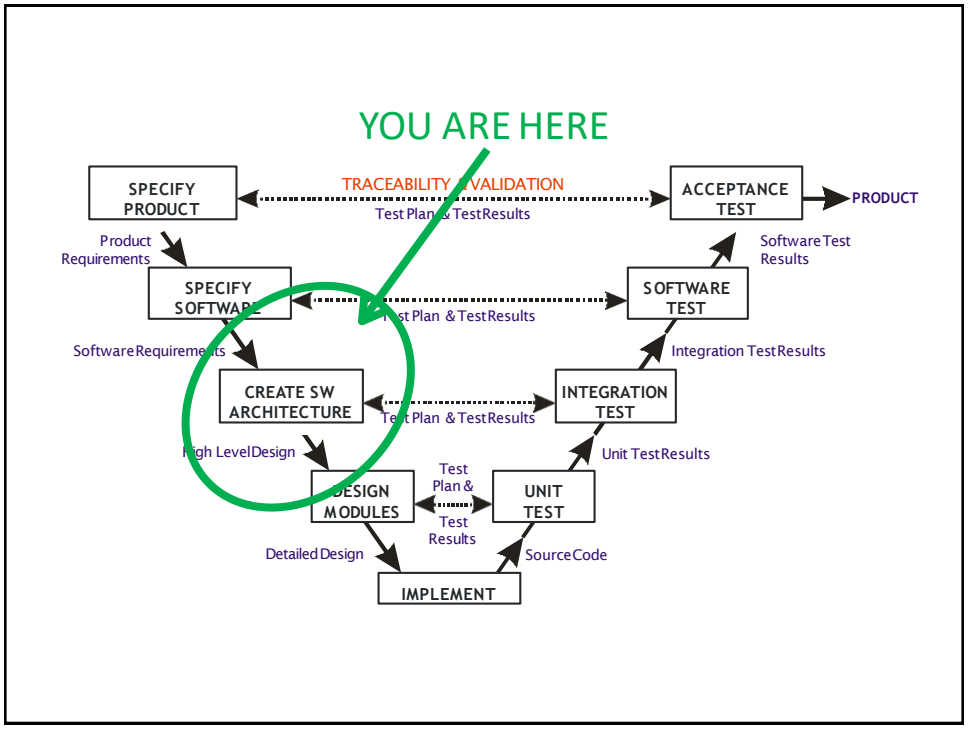
Lecture 5: Embedded Software Architecture

Seyed-Hosein Attarzadeh-Niaki

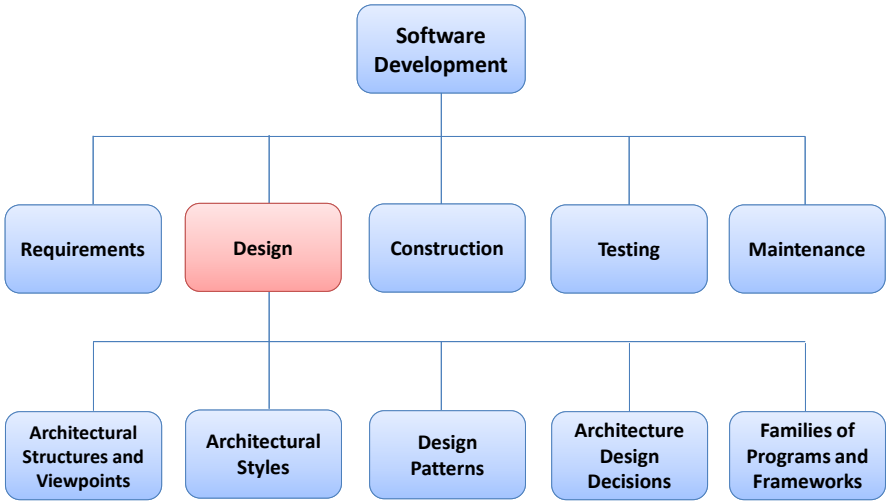
Some Slides from Philip Koopman, Jacob Beningo, and Marilyn Wolf

Review

- Modeling physical dynamics
- Actor-based modeling of continuous-time systems
- Control systems



Software Development Process



IEEE Computer Society SWEBOK v3.0
http://www.computer.org/portal/web/swebok/swebokv3

What is a Software Architecture?

- IEEE 1471
  - Architecture is the fundamental organization of a system embodied in its components, their relationship to each other, and to the environment, and the principles guiding its design and evolution.
- High Level Structure
  - No implementation details!
  - Language agnostic

System Architecture Design

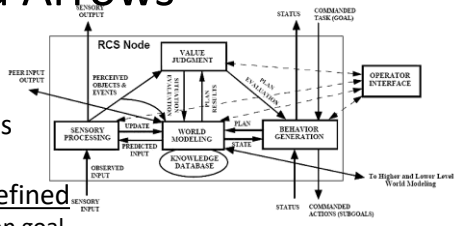
- High Level Design (HLD) = architecture (nouns) + requirements (verbs)

- What major components go satisfying the specification?
- *Hardware* components
  - CPUs, peripherals, etc.
- *Software* components
  - major programs and their operations.
- Must take into account functional and extra-functional specifications.

➢ Architecture Description Languages (ADL)
  - Box-and-line informal drawings
  - Formal architecture description languages
    - AADL, EAST-ADL, EADL, etc.
  - UML-based notations
    - Use case, class, object, component, deployment, sequence, collaboration, statechart, and activity diagrams

Simplified Architecture: Boxes and Arrows

- Software architecture shows the big picture
  - Boxes: software modules/objects
  - Arrows: interfaces
  - Box and arrow semantics well-defined
    - Meaning of box/arrow depends on goal
  - Components all on a single page
    - Nesting of diagrams is OK

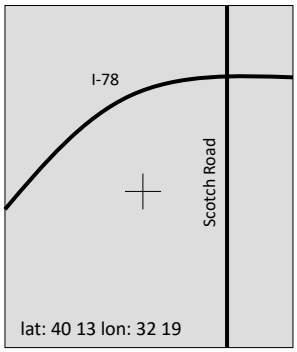


https://goo.gl/WnciF3

- Many different architecture diagrams are possible, such as
  - *Software architecture* (components and data flow types)
  - *Hardware architecture* with software allocation
  - *Controls architecture* showing hierarchical control
  - *Call graph* showing run-time hierarchy

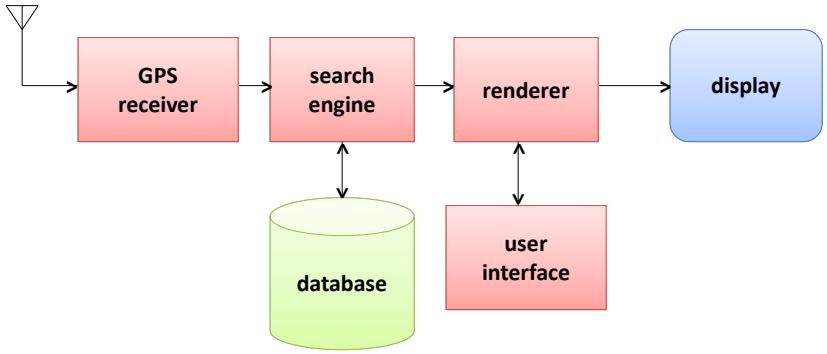
Example: GPS Moving Map Requirements

- Moving map obtains position from GPS, paints map from local database.

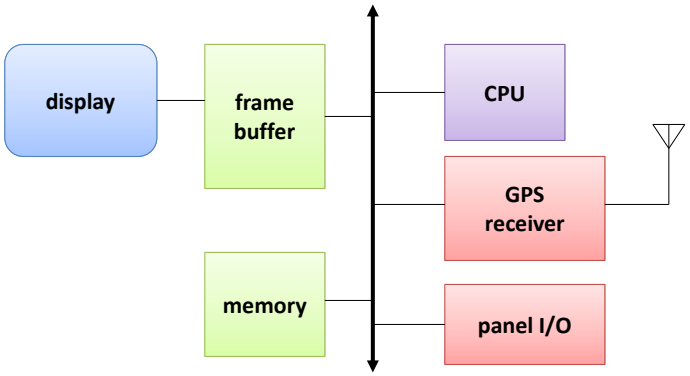


- Functionality: For automotive use. Show major roads and landmarks.
- User interface: At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- Performance: Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- Cost: $120 street price = approx. $30 cost of goods sold.
- Physical size/weight: Should fit in hand.
- Power consumption: Should run for 8 hours on four AA batteries.

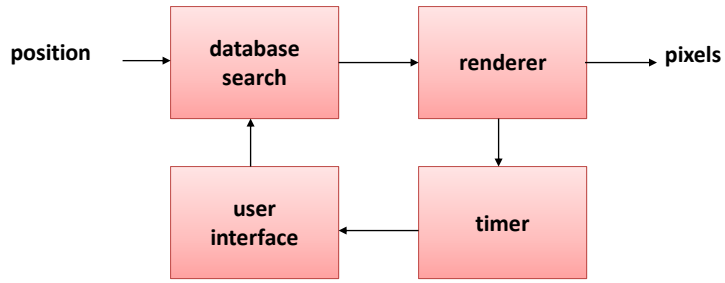
GPS Moving Map Block Diagram (Dataflow Architecture)



GPS Moving Map Hardware Architecture



GPS Moving Map Software Architecture



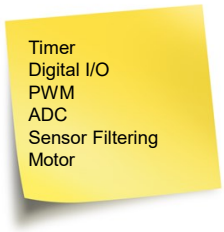
What about the *interfaces*?

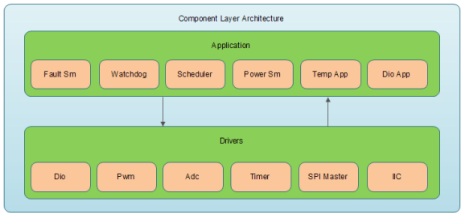
Software Architecture Design Steps

- Step 1 – Identify system components
  - Drivers
    - Microcontroller Peripherals
    - External Sensors
  - Application
    - Driving a motor
    - Filtering sensor data
  - Configuration
    - How peripherals and sensors need to be setup

Software Architecture Design Steps

- Step 2 – Understand component relationships
  - Which components interact?
  - What layer does the component belong in?
  - Create a component layer architecture



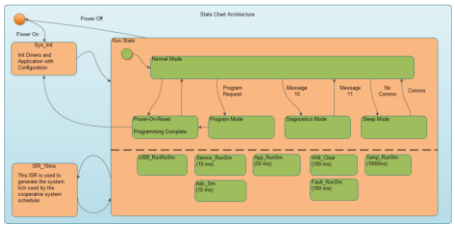


Software Architecture Design Steps

- Step 3 – Define the Environmental Factors
  - Hard Real-Time Constraints
  - Soft Real-Time Constraints
  - Sensors and hardware interfaces
  - System Timing
  - Human interactions
  - Identify synchronous events
  - Identify asynchronous events

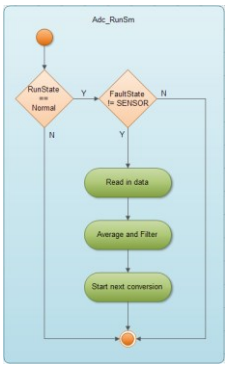
Software Architecture Design Steps

- Step 4 – Create a High Level Architecture
  - Primary system states
  - Interrupts
  - System Tasks
  - System State Machines
  - System Events
  - Concurrent States
  - Entry/Exit Points



Software Architecture Design Steps

- Step 5 – Implement Flowcharts and Sequences
  - The devil is in the details
  - Expand the state machines behavior
  - Identify *communication sequences*
  - Simulate the flowchart behavior
  - Identify high risk areas
  - Refactor and simplify
  - Keep it language independent



Software Architecture Design Steps

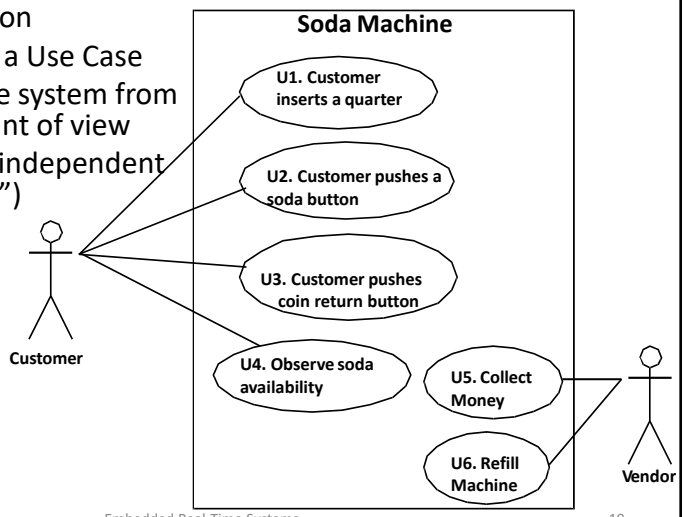
- Step 6 – Review the Architecture
  - Let the architecture sit for a day or two
    - Forget and come back to it with fresh eyes
  - Review with a group of peers
  - Flush out missing components, complications
  - Identify potential areas of risk
  - Verify system requirements
  - Update and refactor
  - Always remember to keep it simple!

Example System: Soda Vending Machine

- High Level Requirements
  - Make it work like a real vending machine
- Simplification
  - Sodas cost some number of quarters
  - All other coins are rejected (invisible to your control system)
- Assume a Distributed System per given diagram
  - Processor for each button, coin return controller, vending controller
  - You get the message dictionary and most of the requirements specification (the “Architecture”)

UML Use Cases for Requirements Development

- Actor is a person
- Actor initiates a Use Case
- Represents the system from the actor's point of view
- Use cases are independent ("transactions")

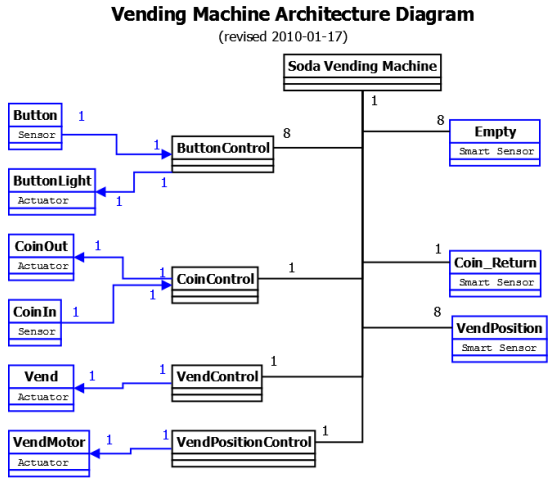


System-Level Text Requirements

- Goal: implement a soda vending machine
  - R1. Pushing a button **shall** vend a soda of the type corresponding to that button.
  - R2. The machine **shall** permanently retain exactly SODACOST coins for each can of soda vended.
  - R3. Coin return **shall** return all deposited coins since the last vend cycle.
  - R4. The machine **shall** return all deposited money in excess of SODACOST coins before a vend cycle.
  - R5. The machine **shall** flash the light for a selected item while vending is in progress to indicate acceptance of a selection to the buyer.
  - R6. The machine **shall** illuminate the light for any out-of-stock item
- Assume a Fully Distributed System
  - Processor for each button, coin return controller, vending controller

Traceability: UML & Text Requirements

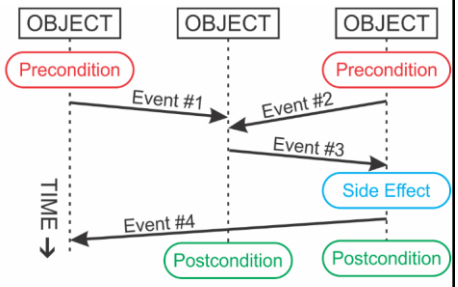
	Text Requirements					
Use Cases	R1	R2	R3	R4	R5	R6
U1. Customer inserts a quarter				X		
U2. Customer pushes a soda button	X				X	
U3. Customer pushes coin return button			X			
U4. Observe soda availability						X
U5. Collect money		X				
U6. Refill machine		X				X



Legend: Blue = physical objects / Black = microcontrollers with software

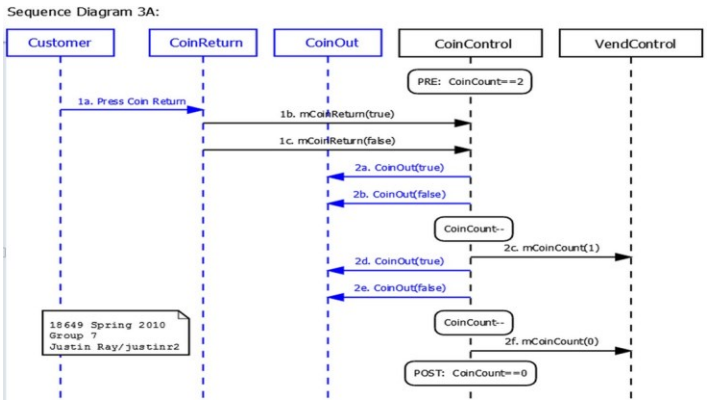
Sequence Diagram (SD) as A High-Level Design (HLD) Notation

- SD construction
  - Each object has a time column extending downward
  - Arcs are interactions between objects
- Each SD shows a scenario
  - Top ovals are *preconditions*
  - Middle ovals are *side effects*
  - Bottom ovals are *postconditions*



- SD is a *partial* behavioral description for objects
  - Generally, each object participates in multiple SDs; each SD only has some objects
  - The set of all SDs forms the HLD for all objects in the system

Example

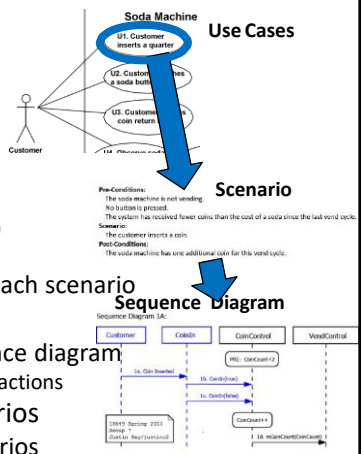


Legend: Blue = physical objects / Black = microcontrollers with software

PRE = precondition / POST = postcondition / other ovals are side effects

Use Cases to Sequence Diagrams

- Use Case diagram – types of interactions
  - System has multiple use cases
  - Example: Use Case #1: Insert a coin
- Scenario – a specific variant of a use case
  - Each use case has one or more scenarios
    - Scenario 1.1: insert coin to add money
    - Scenario 1.2: insert excess coin (too many inserted)
    - Scenario 1.3: … some other situation…
  - Interactions between objects are different for each scenario
- Sequence Diagram – a specific scenario design
  - For our purposes each scenario has one sequence diagram
    - Sequence diagrams 1.1, 1.2, 1.3 show specific interactions
- Statechart – design that incorporates all scenarios
  - One StateChart per object, addressing all scenarios

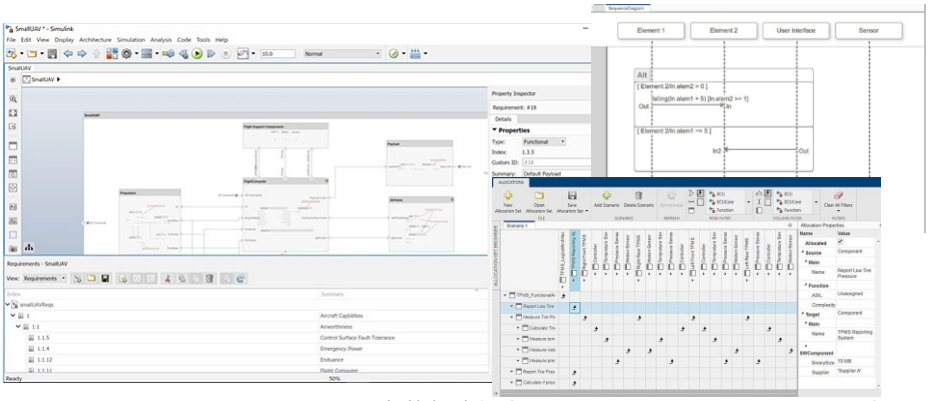


Embedded Software Architectures

- Common Embedded Software Patterns
  - Polling
  - Interrupt triggered (blocking)
  - Interrupt triggered (non-blocking)
  - Event driven
  - RTOS
  - Energy Aware

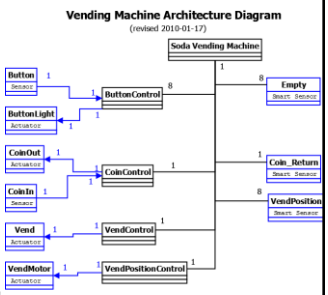
MATLAB System Composer

- Compose a hierarchical block diagram of the system/software architecture
- Specify and refine interfaces on ports
- Requirements allocation and traceability
- Behavioral modeling using block diagrams, activity diagram, SDs, etc.
- Architecture allocation: link between architecture models



High Level Design (HLD) Best Practices

- HLD includes
  - One or more architecture diagrams
    - Defines all components & interfaces
    - HW arch., SW arch., Network arch., …
  - Sequence Diagrams
    - Both nominal and off-nominal interactions
    - HLD must co-evolve with requirements
  - Need both nouns + verbs to define a system!
- High Level Design pitfalls
  - Diagrams that leave out interactions
  - Boxes and arrows don't have well defined meanings
  - HLD that bleeds into detailed design information
    - Should have separate Detailed Design per component



https://users.ece.cmu.edu/~koopman/ece649/project/sodamachine/index.html

Next Lecture

- Discrete systems
- State space
- Finite-state machines
  - Deterministic FSMs
  - Non-deterministic FSMs
- Read chapter 3 of LeeSeshia