Lecture 2: Developing Embedded Real-time Systems

Seyed-Hosein Attarzadeh-Niaki

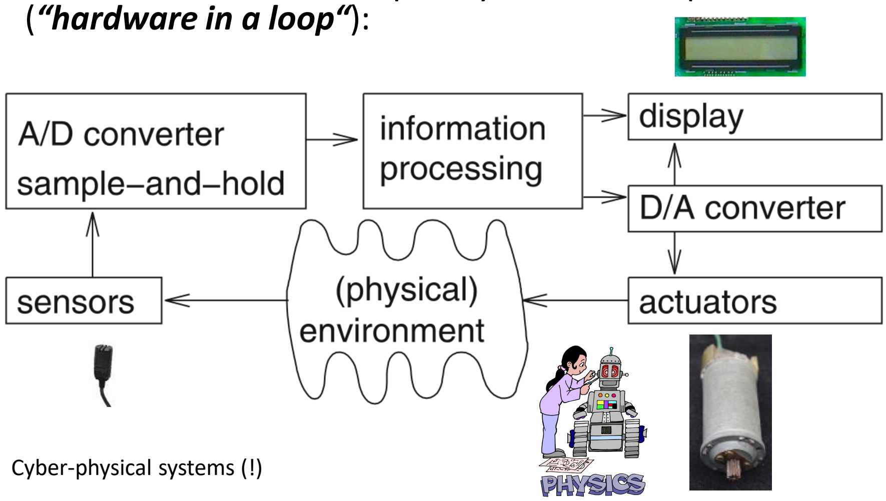
Some Slides from Peter Marwedel and Edward Lee, and Philip Koopman

Review

- What is an Embedded/Cyber-Physical System?
- Why they are important?
- How to design them?

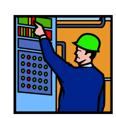
CPS & ES Hardware

- CPS & ES hardware is frequently used in a loop (***"hardware in a loop"***):



Cyber-physical systems (!)

Reactive & Hybrid Systems



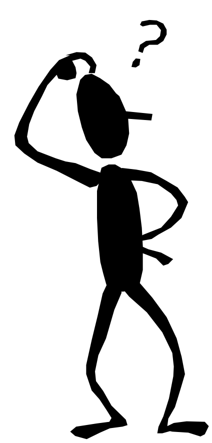
- Typically, CPS are **reactive systems**:
  - "**A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment**" [Bergé, 1995]
  - Behavior depends on input **and current state**.
    ☞ automata model appropriate,
    model of computable functions inappropriate.
- **Hybrid systems** (analog + digital parts).

Challenges for implementation in hardware

- Early embedded systems frequently implemented in hardware (boards)
- Mask cost for specialized application specific integrated circuits (ASICs) becomes very expensive
  (M$ range, technology-dependent)
- Lack of flexibility (changing standards).
- Trend towards implementation in software (or possibly FPGAs)

Challenges for implementation in software

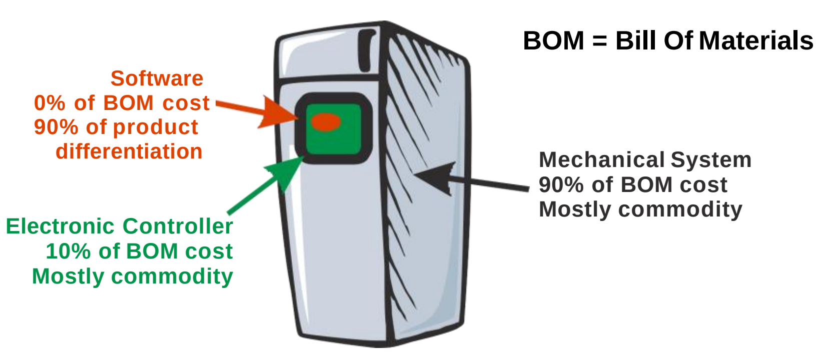
If CPS/ES will be implemented mostly in software, then why don't we just use what software engineers have come up with?



Challenges for CPS/ES Software

- Dynamic environments
- Capture the required behavior!
- Validate specifications
- Efficient translation of specifications into implementations!
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)

Act As If Your Products Live Or Die By Their Software



Software Complexity is a Challenge

Software in a TV set

- Source 1*:

Year	Size
1965	0
1979	1 kB
1990	64 kB
2000	2 MB

- Source 2°: 10x per 6-7 years

Year	Size
1986	10 KB
1992	100 kB
1998	1 MB
2008	15 MB

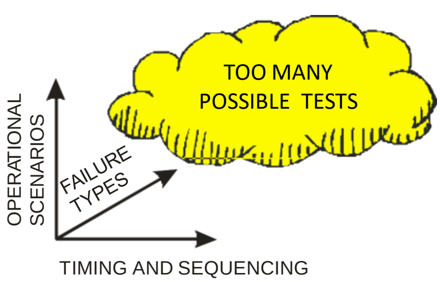
- ☞ Exponential increase in software complexity
- *... > 70% of the development cost for complex systems such as automotive electronics and communication systems are due to software development*
[A. Sangiovanni-Vincentelli, 1999]

* Rob van Ommering, COPA Tutorial, as cited by: Gerrit Müller: Opportunities and challenges in embedded systems, *Eindhoven Embedded Systems Institute*, 2004

° R. Kommeren, P. Parviainen: Philips experiences in global distributed software development, *Empir Software Eng.* (2007) 12:647-660

Product Testing Won't Find All Bugs

- Testing bad software simply makes it less bad
  - Testing cannot produce good software all on its own

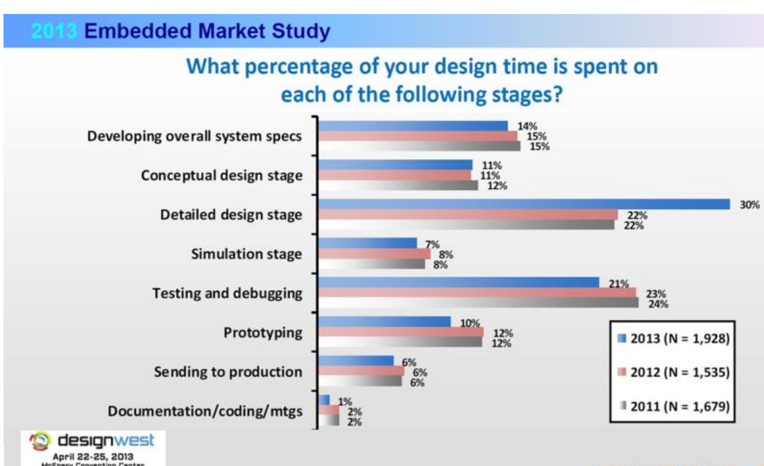


- One third of faults take more than 5000 years to manifest
Adams, N.E., "Optimizing preventive service of software product," IBM Journal of Research and Development, 28(1), p. 2-14, 1984. (Table 2, pg. 9, 60 kmonth column)

- Your customers will regularly experience bugs that you will not see during testing
- For most products, you can't even test 5000 years

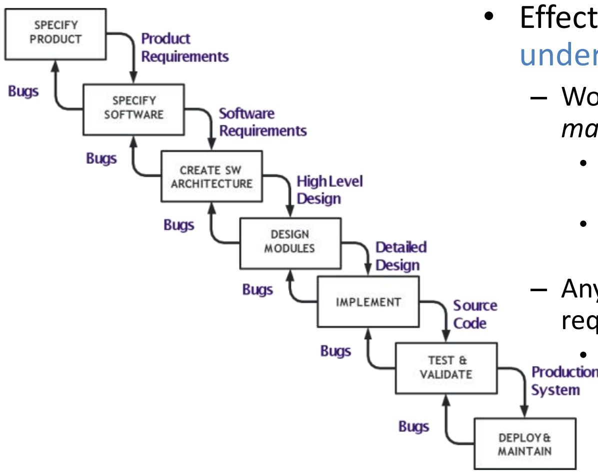
SOFTWARE DEVELOPMENT PROCESS

Coding is Essentially 0% of Creating Software



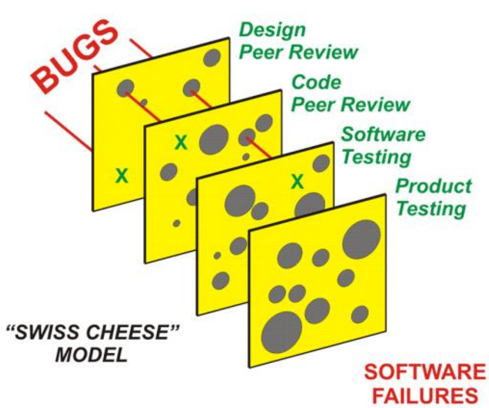
http://e.ubmelectronics.com/2013EmbeddedStudy/index.html

Old-School Waterfall Development Cycle



- Effective for well understood domains
  - Works best if you *don't make many big mistakes*
    - Variations on existing systems
    - Expensive to fix things that escape to test steps
  - Any problem encountered requires backtracking
    - Note: original waterfall paper had these backward arrows! It was never just a unidirectional process

How to Get High Quality Software



- Product Testing
  - Late & Expensive
  - Many field escapes
- Software Testing
  - Unit & Integration test
- Code Peer Review
  - Earlier & Cheaper
- Design Peer Review
  - Earlier & Cheaper

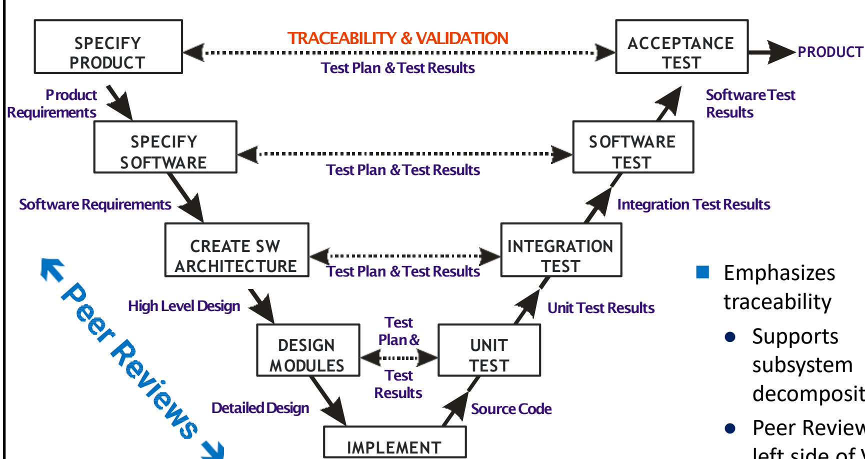
What We've Learned in 50+ Years of Software

- Dividing up into subsystems is critical
  - Bad architecture will doom a project
- Process formality is a good investment
  - Traceability, formal reviews, etc.
  - Skipping steps costs more in the end
- Requirements change
  - Suggests using an iterated approach
- Finding bugs early is important
  - Traceability from high to low levels
  - Layered testing
  - Peer reviews, especially on left side of V

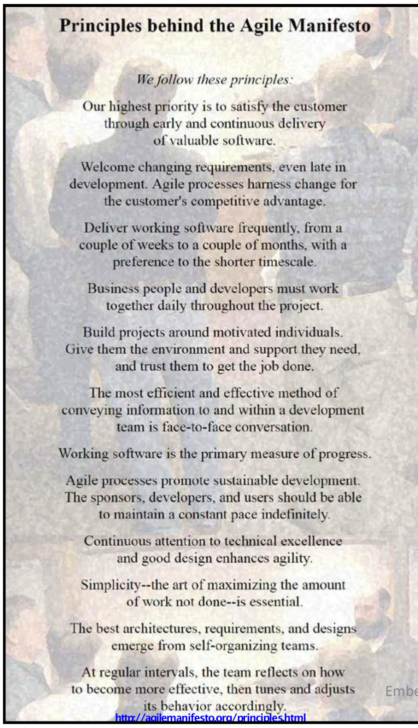
- **If the second half of the project is "debugging" that must mean the first half is "bugging"**
  - **Jack Ganssle**

http://www.ganssle.com/rants/ontesting.htm (paraphrase)

V (or "Vee") Development Cycle



- Emphasizes traceability
  - Supports subsystem decomposition
  - Peer Reviews on left side of V



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://agilemanifesto.org/principles.html

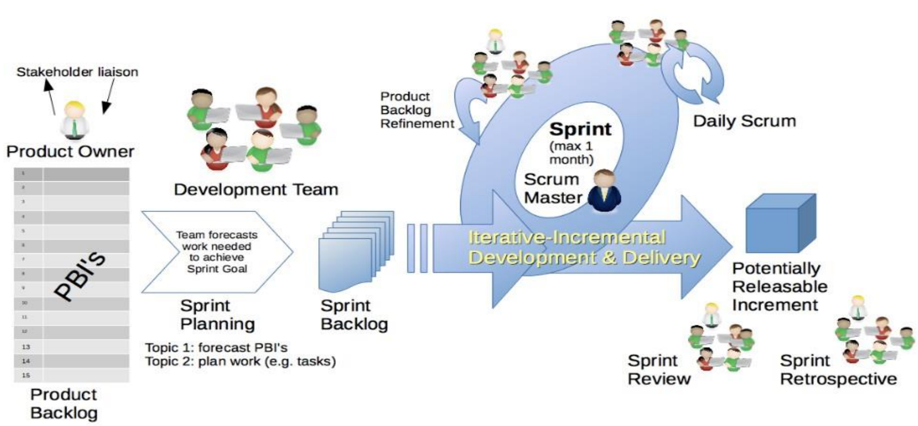
Agile Methods

- **Agile generally values:**
  - **Individuals and Interactions over processes and tools**
  - **Working Software over comprehensive documentation**
  - **Customer Collaboration over contract negotiation**
  - **Responding to Change over following a plan**

- **Example: Scrum**
  - **Daily "stand up" ("scrum") meetings for face-to-face collaboration**
  - **2-4 week long sprints to incrementally add functionality**
    - **Each sprint implements items from a backlog**
    - **Demo at end of sprint; theoretically a shippable product**
  - **User stories serve as requirements**
  - **Scrum challenges**
    - **Geographically split teams with informal communication**
    - **External dependencies (e.g., other parts of system change)**
    - **No time for extensive testing, especially embedded hardware**

Scrum Process Example

- Heavy on implicit knowledge
  - Can you find: requirements, design, test plan, acceptance test?



Agile Methods + Embedded (?)

- Significant benefit is that it makes (good) developers happier
  - If done well can help with evolving requirements
  - But, you need to manage and moderate the risks
- **Issue**: "Agile" is not just cowboy coding
  - Undefined, undisciplined processes are bad news
  - Yes, Agile teams should follow a rigorously defined process
- **Issue**: "No-paper" Agile unsuitable for long-lived systems
  - Implicit knowledge is efficient, but evaporates with the team
  - 10+ year old undocumented legacy systems are a nightmare
- **Issue**: Agile assumes 100% automated acceptance test
  - 100% automated system test is often impractical for physical interfaces
  - Often implicitly assumes that defect escapes are low cost because a new version is 2-4 weeks away
- **Issue**: Agile typically doesn't have independent process monitoring (SQA)
  - Software Quality Assurance (SQA) tells you if your process is working
  - Agile teams may be dysfunctional and have no idea this is happening
    - Or they may be fine – but who knows if they are really healthy or not?

When is Agile a Good Fit?

Source: Boehm & Turner 2004, Balancing Agility and Discipline

Agile

- Small teams; small products
- "Everyday" software quality
- Fast requirements change
- High-skill experts throughout project
  - Including life-cycle maintenance
- Developers can handle being empowered; usually senior

Plan-Driven (traditional)

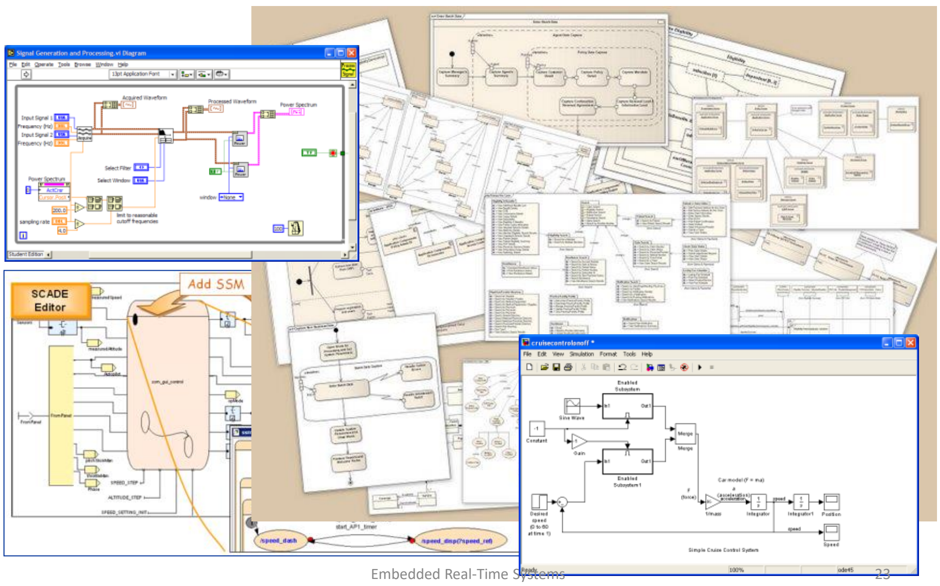
- Large teams; large products
- Mission-critical products
- Stable requirements
- High skill primarily in design phase
  - Major versions require expert design
- Most developers are not empowered; usually junior

Best Practices For Software Process

- Follow a defined process
  - Must include all aspects shown on Vee
    - And SQA, Peer Reviews
  - It's OK to rename and reorganize steps
    - All the steps have to get done
    - Common to see "AgileFall" etc.
    - Also common to see bad process dressed up with the latest buzzwords
- Software Process Pitfalls
  - Skipping steps to get to testing faster means more bugs in test
    - Finding bugs is more expensive in testing
  - Using the wrong process for the wrong purpose
    - 3-Week product life and 30 year product life are different situations

MODEL-BASED DESIGN

Focus on Models



Motivation for Considering Specs & Models

- Why considering specs and models in detail?
  – If something is wrong with the specs, then it will be difficult to get the design right, potentially wasting a lot of time.
- Typically, we work with models of the system under design (SUD)
- What is a model anyway?

Models

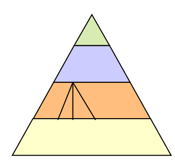
Definition: *A model is a simplification of another entity, which can be a physical thing or another model. The model contains exactly those characteristics and properties of the modeled entity that are relevant for a given task. A model is minimal with respect to a task if it does not contain any other characteristics than those relevant for the task*.

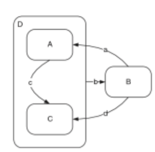
[Jantsch, 2004]

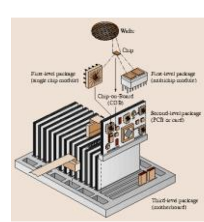
Which requirements do we have for our models?

Model Hierarchy

- Humans not capable to understand systems containing more than ~5 objects.
  Most actual systems require more objects
  ☞ Hierarchy (+ abstraction)
  - Behavioral hierarchy
    Examples: states, processes, procedures.
  - Structural hierarchy
    Examples: processors, racks, printed circuit boards

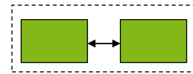






Component-Based Design

- Systems must be designed from components
  - Especially for architecture design



- Must be "easy" to derive behavior from behavior of subsystems

- Concurrency
- Synchronization and communication

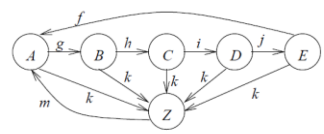
Time in Models

- Timing behavior
  Essential for embedded and cy-phy systems!
  - Additional information (periods, dependences, scenarios, use cases) welcome
  - Also, the speed of the underlying platform must be known
  - Far-reaching consequences for design processes!

"The lack of timing in the core abstraction (of computer science) *is a flaw, from the perspective of embedded software"* [Lee, 2005]

Modeling Support for Designing Reactive Systems

- **State-oriented behavior**
  Required for reactive systems; classical automata insufficient.



- **Event-handling**
  (external or internal events)
- **Exception-oriented behavior**
  Not acceptable to describe exceptions for every state

Other Modeling Requirements

- **Executability** (not only algebraic specification)
- **Support for the design of large systems** (☞ OO?)
- **Domain-specific support**
- **Readability**
- **Portability and flexibility**
- **Support for non-standard I/O devices**
- **Extra-functional properties**
- **Support for the design of dependable systems**
- **No obstacles for efficient implementation**
- **Adequate models of computation**
  What does it mean "to compute"?

Problems with classical CS theory and von Neumann (thread) computing

- Even the core … notion of "computable" is at odds with the requirements of embedded software.
- In this notion, useful computation terminates, but termination is undecidable.
- In embedded software, termination is failure, and yet to get predictable timing, sub-computations must decidably terminate.

➢ *What is needed is nearly a reinvention of computer science.*

Edward A. Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Search for non-thread-based, non-von-Neumann MoCs.

Determinacy

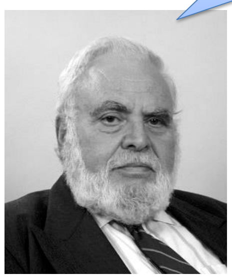
Some of the most valuable models are *deterministic.*

A model is *deterministic* if, given the initial state and the inputs, the model defines exactly one behavior.

Deterministic models have proven extremely valuable in the past.

Engineers often confuse the model with its target

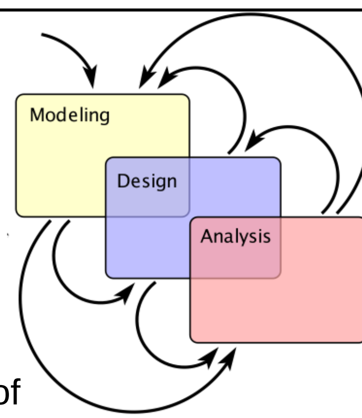
You will never strike oil by drilling through the map!



Solomon Wolf Golomb

But this does not in any way diminish the value of a map!

Modeling, Design, Analysis An Iterative Process



Modeling is the process of gaining a deeper understanding of a system through imitation. Models express **what** a system does or should do.

Design is the structured creation of artifacts.
It specifies **how** a system does what it does.

Analysis is the process of gaining a deeper understanding of a system through dissection.
It specifies **why** a system does what it does (or fails to do what a model says it should do).

Next Lecture

- CPS and ES requirements
  - Functional requirements
  - Extra-functional requirements
- Requirement analysis and specifications