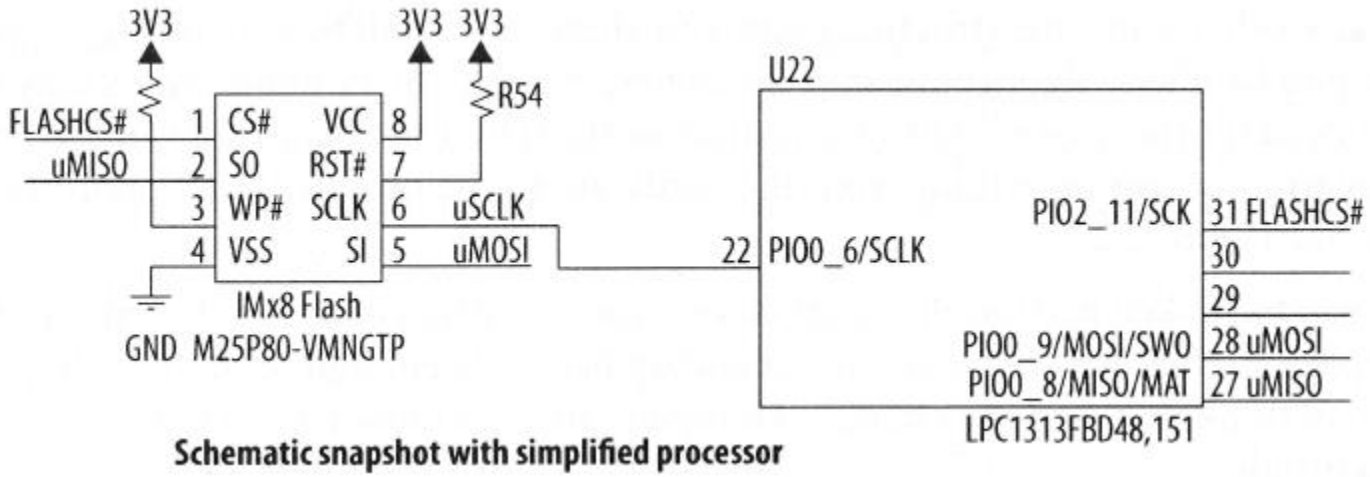
Föreläsning 4 - mjukvara

- Erfarenheter från Pingpong-exemplet
  - Testdriven utveckling
  - Moduluppbyggnad
  - Tillståndsdiagram
  - Kommentarer
  - Vettiga namn på funktioner och variabler
- Arkitektur för programvaran
- Större uppgift i Inbyggda system

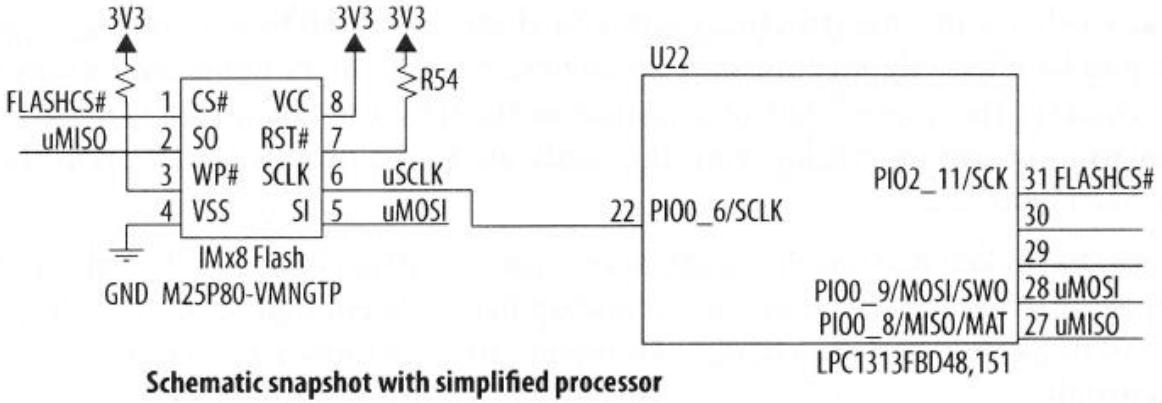
Creating a System Architecture*



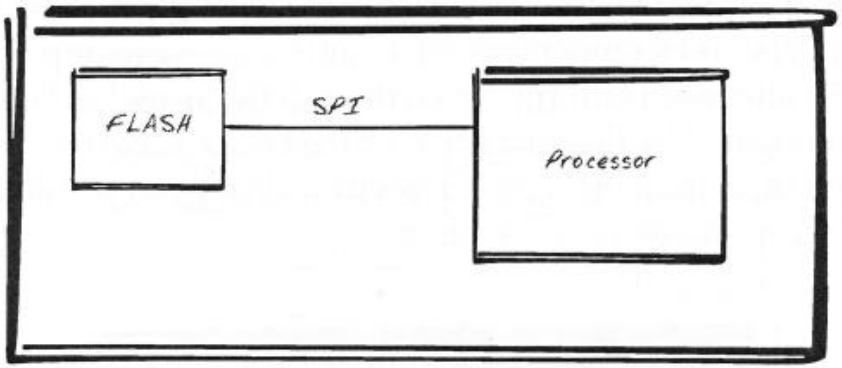
Schematic snapshot with simplified processor

* Source: White, Making Embedded Systems

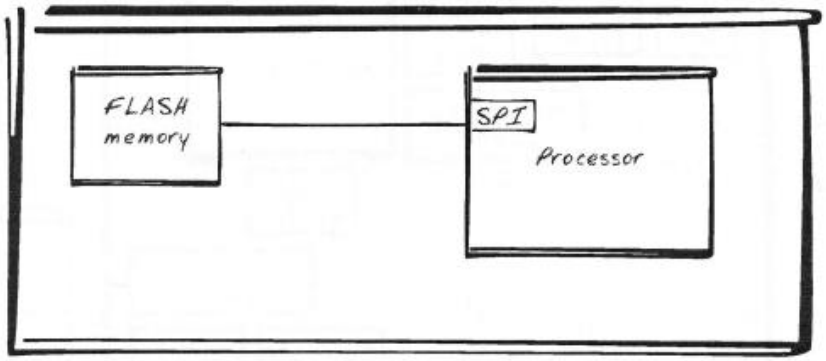
Block Diagram



Schematic snapshot with simplified processor



Hardware block diagram

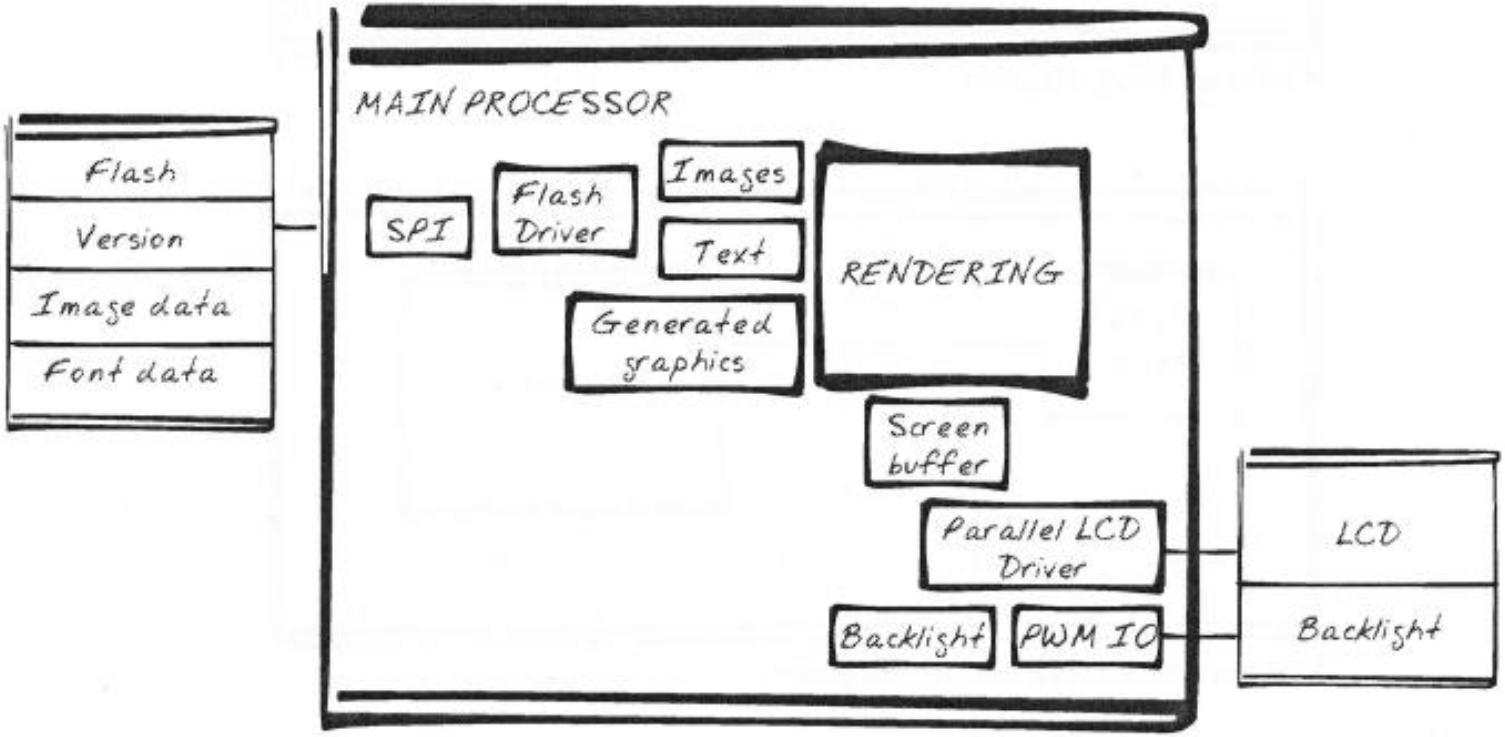


Software architecture block diagram

Hardware Block Diagram

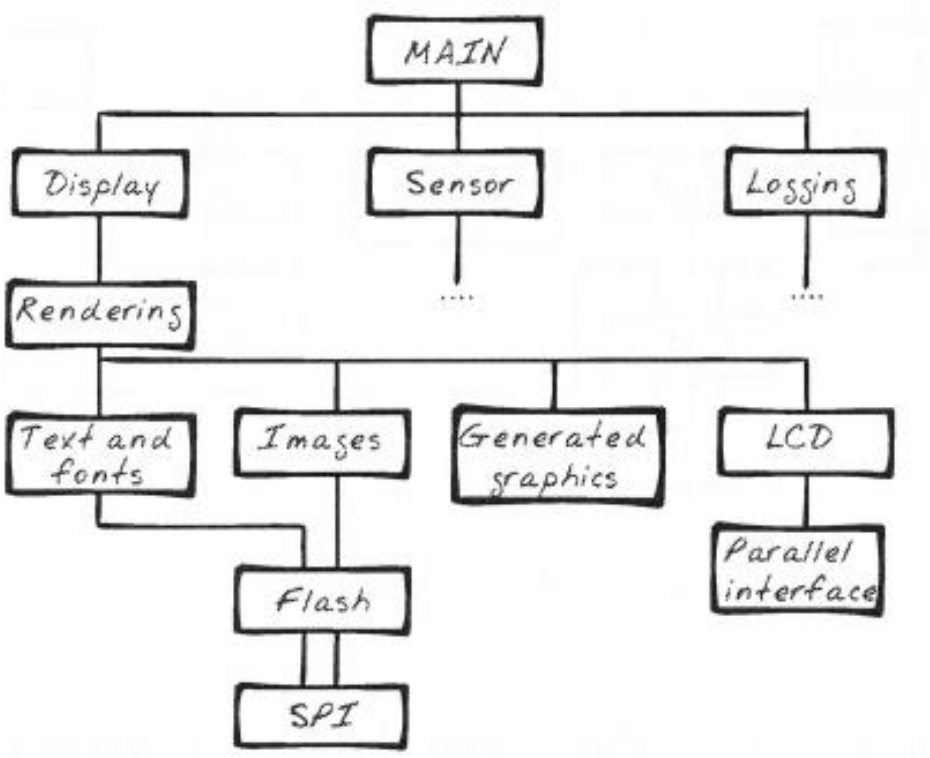
Software Architecture Block Diagram

Software Block Diagram



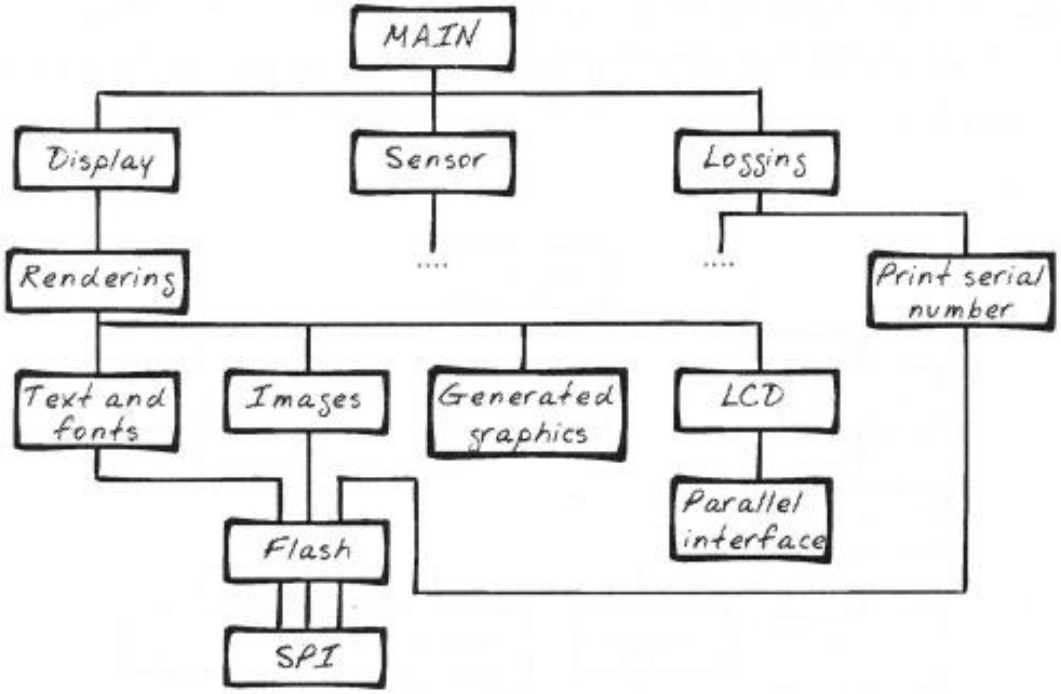
Sketch the system , try to figure out as many boxes as possible, combine them later
Looking at various views may show you some hidden spots with critical bottlenecks, poorly understood specifications or failure to implement on intended platform
Identify tricky modules and see a path to a good solution.

Hierarchy of Control



A hierarchical view of the software. Main is highest level.
Fill in next levels with algoritm-related objects.
It shows discrete components and which components call others

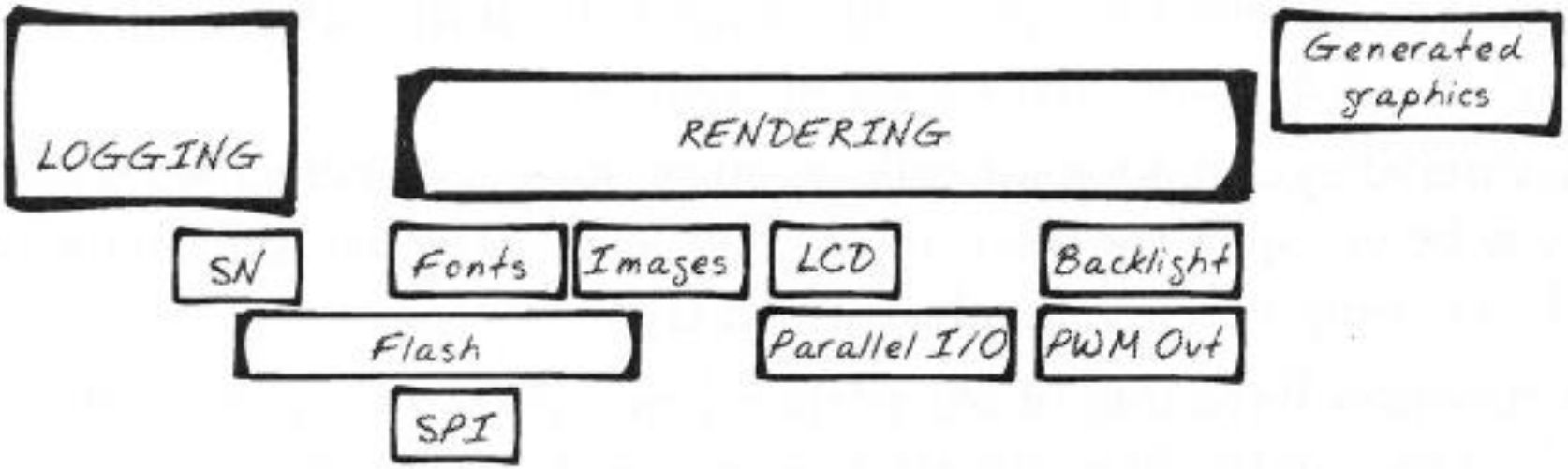
Hierarchy of Control, shared resource



Each time something that this is added, some little piece where you are using A and B and have to consider a potential interaction with C, the system becomes a little less robust.

Shared resources may cause pains in the design, implementation and maintance phases of the project.

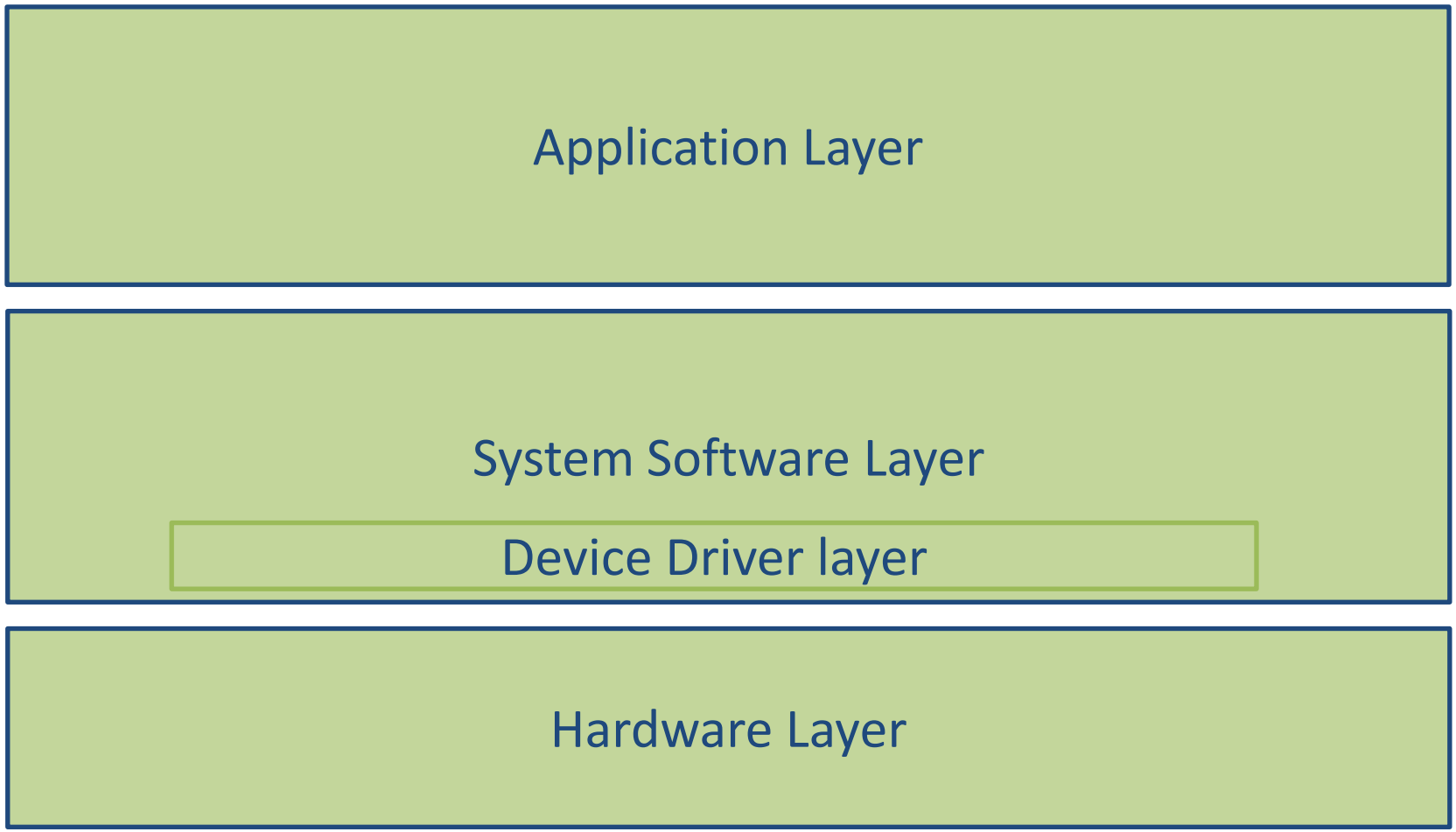
Layered Software Architecture Diagram



Represent objects by their estimated size. Let the size reflect the complexity.
Start at the bottom of the page and draw boxes for the the things that go off the processor (communication boxes).
Add to the diagram the items that use the lowest layer.
Each object that uses something below it should touch all of the things it uses.

This shows you where the layers in your code are.

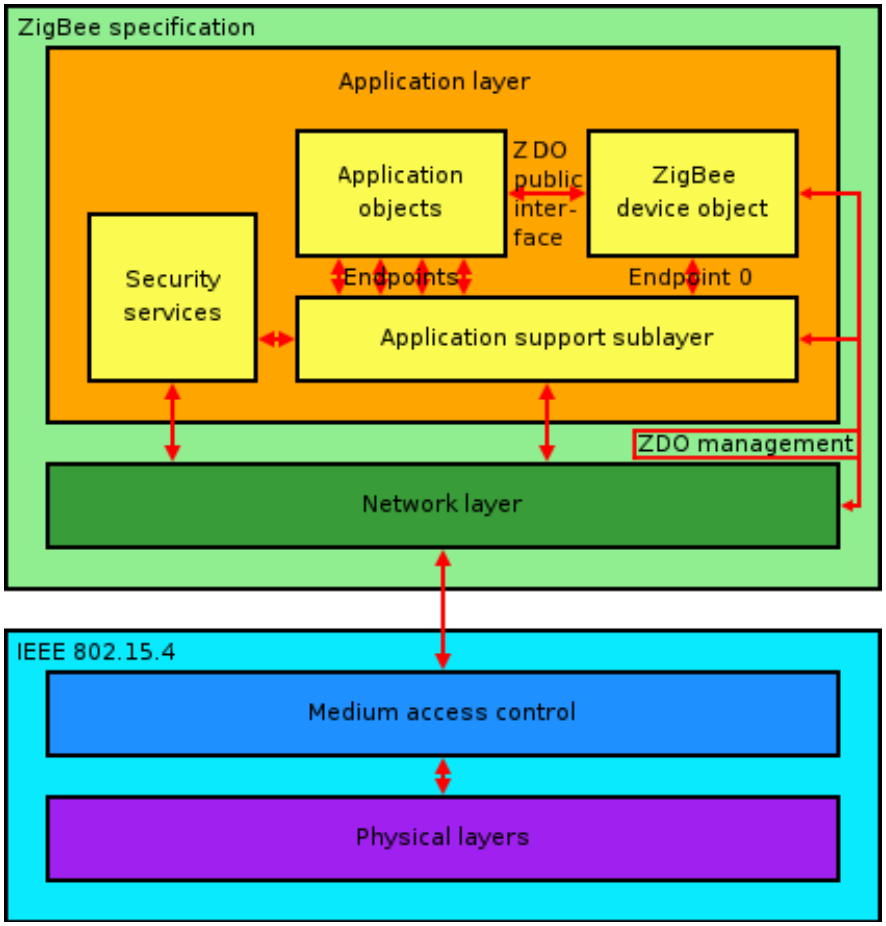
Code layers


Application Layer
System Software Layer
Device Driver layer
Hardware Layer

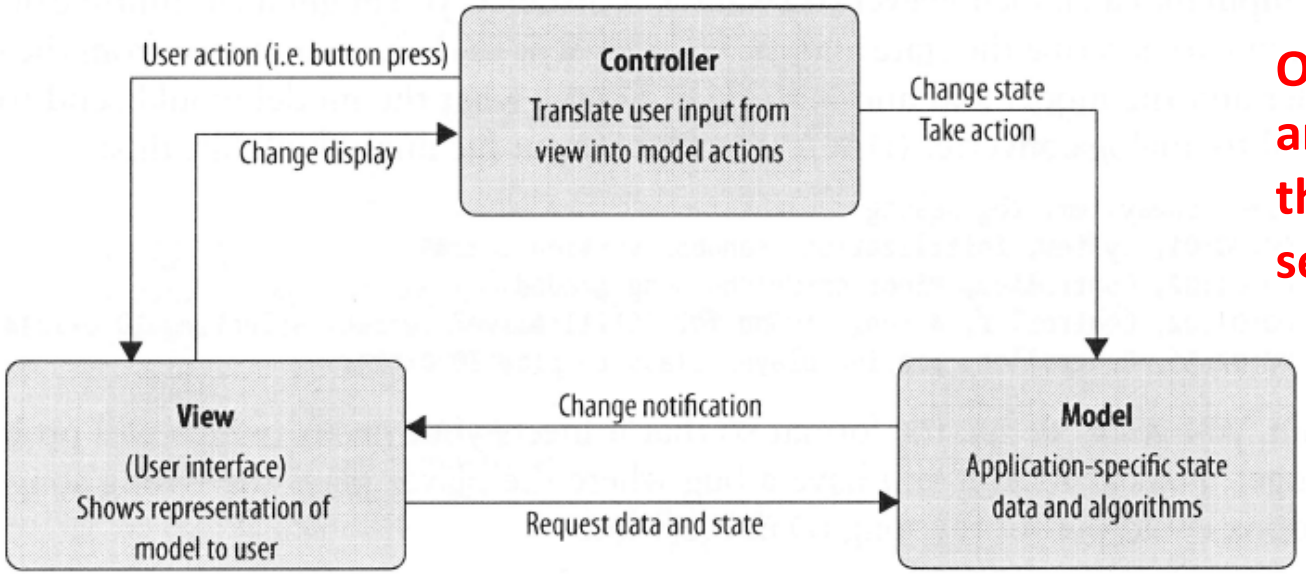
OSI model

OSI Model			
	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication, managing sessions between applications
	Segments	4. Transport	Reliable delivery of packets between points on a network.
Media layers	Packet/Datagram	3. Network	Addressing, routing and (not necessarily reliable) delivery of datagrams between points on a network.
	Bit/Frame	2. Data link	A reliable direct point-to-point data connection.
	Bit	1. Physical	A (not necessarily reliable) direct point-to-point data connection.

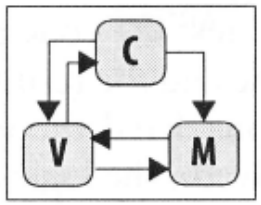
Exempel ZigBee stack



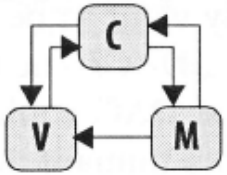
User interface, display GUI* Model-View-Controller, algoritm boxes



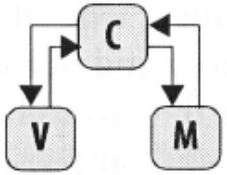
One goal of good architecture is to keep the algoritm as segregated as possible



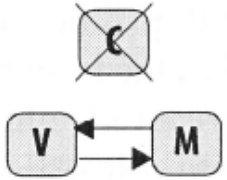
Shown here



Model receives data only from controller



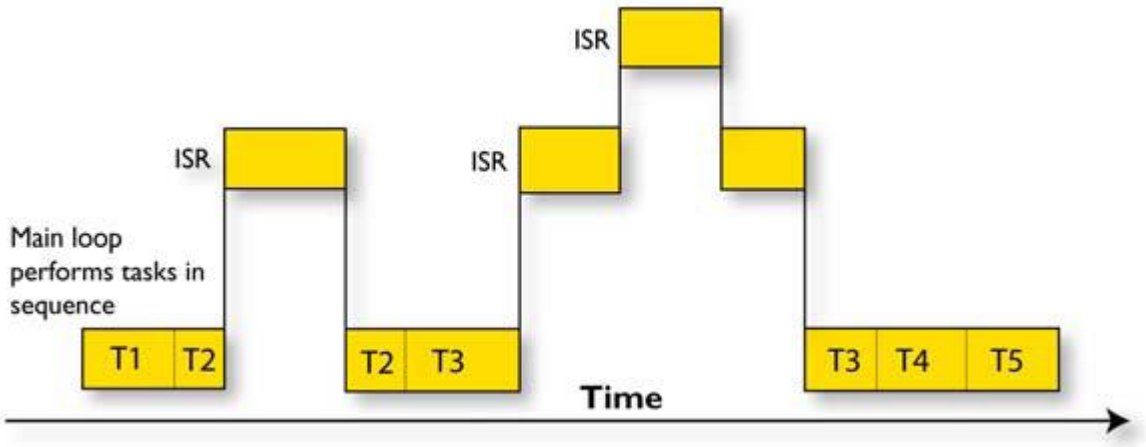
Controller is translator



Model-View pattern

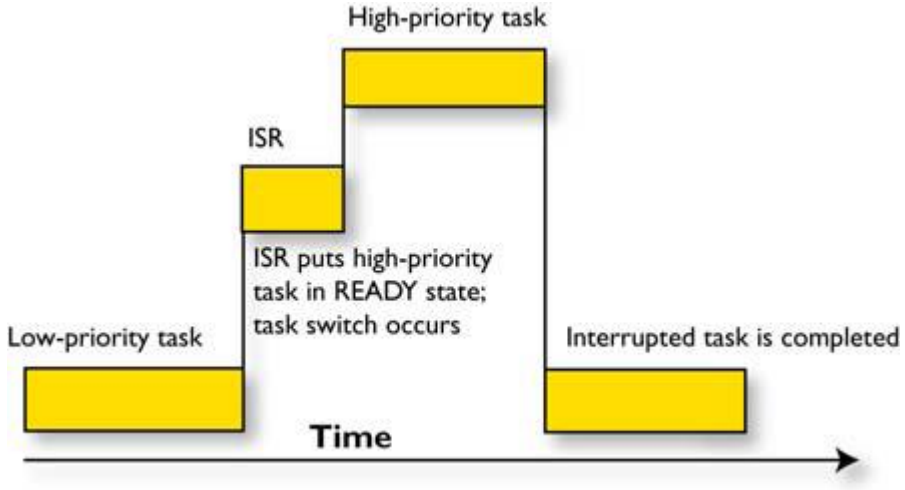
*GUI = Graphical User Interface

Superloop system (single task system)



- Tasks executes in sequence
- No realtime OS, real time not important
- Using interrupt for time critical tasks
- Hard to use in big systems

Preemptive multitasking system



RTOS = Real Time Operating System

- Multiple tasks run ”simultaneously”
- Tasks are scheduled so the RTOS can activate or deactivate tasks
- Timer interrupt to change tasks

Versionshantering

- Under utveckling så ändras filer hela tiden och inte alltid till det bättre
- Vi behöver ett system för att hålla ordning på de ändringar som görs
- Nödvändigt när flera jobbar i samma projekt
- I versionshanteringssystemet (VCS = Version Control System) kan du checka in och checka ut kod från en server

Ref: http://en.wikipedia.org/wiki/Revision_control

Programutveckling för eget inbyggt system

- Strukturera, dela upp i moduler
  - Blockschema
- Arkitektur
  - Hierarkist (Hierarchy of Control
  - Lager (Layered Software Architecture Diagram)
- Testning av moduler
- Utveckla varje modul
  - Pseudokod
  - Tillståndsdiagram
  - Flödesschema
  - Skriv kod
- Utveckla applikation

Uppgift

- Har du kanske listat ut uppgiften från de ledtrådar som jag gav i måndagens föreläsning

?

Uppgift: Turtle Graphics

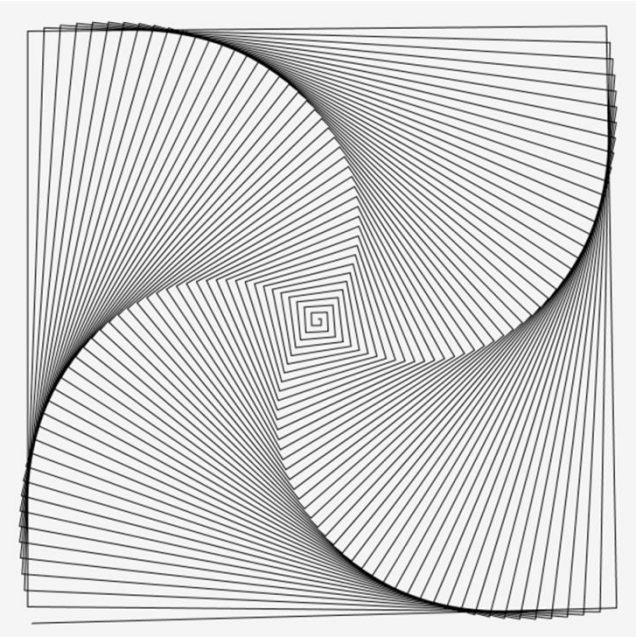
The turtle has three attributes:

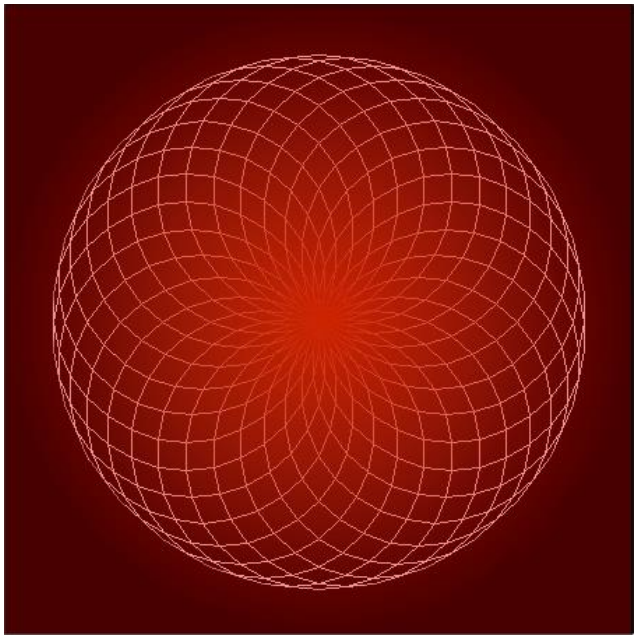
1. a location
2. an orientation
3. a pen, itself having attributes such as color, width, and up versus down.

The turtle moves with commands that are relative to its own position, such as "move forward 10 spaces" and "turn left 90 degrees". The pen carried by the turtle can also be controlled, by enabling it, setting its color, or setting its width.

Källa: http://en.wikipedia.org/wiki/Turtle_graphics

Några exempel





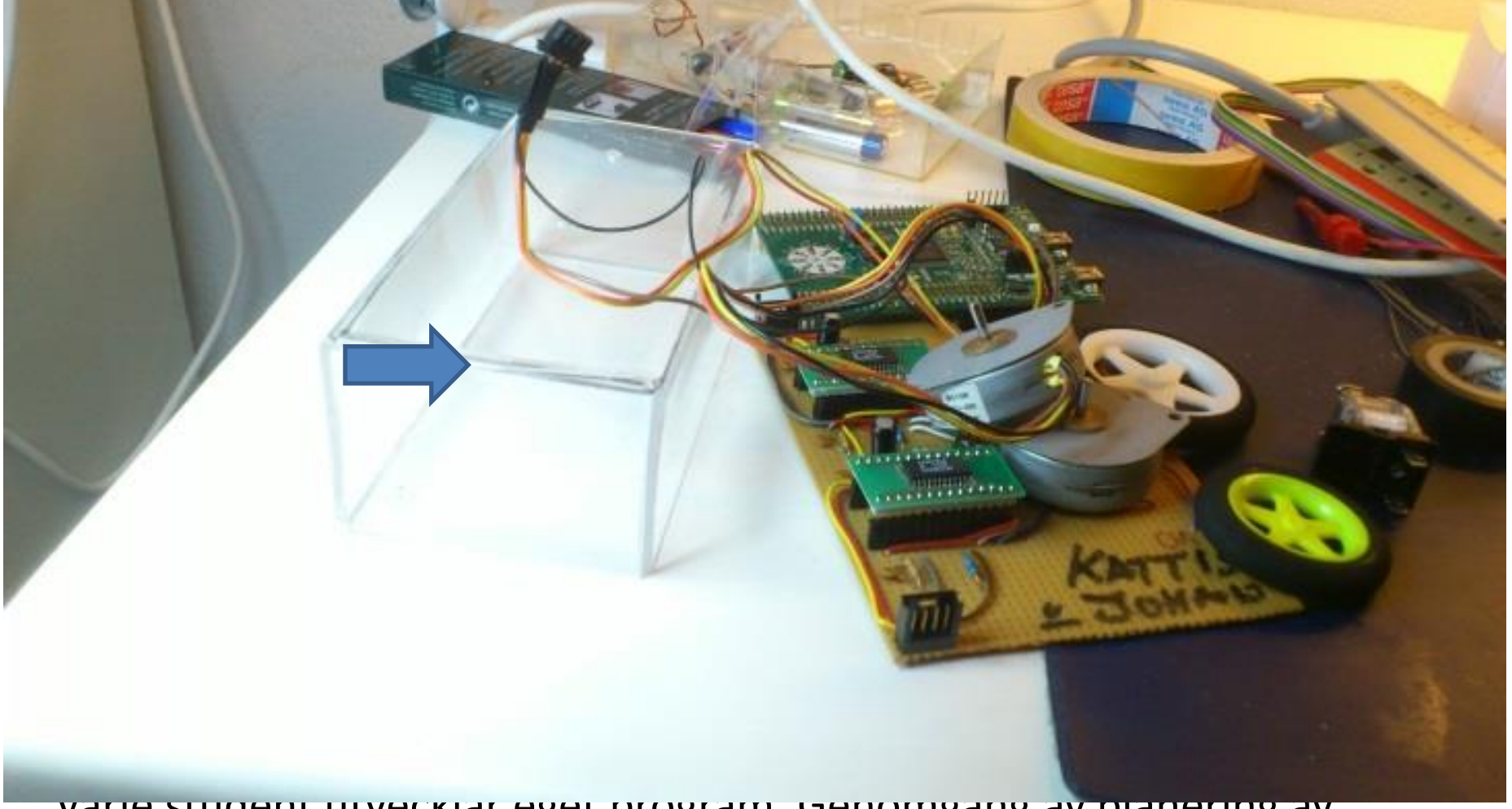
Logo

- Turtle Graphics finns/fanns i programmeringsspråket Logo (1967)
- Vi skall bygga en Turtle Robot!
  - Penna upp, Penna ned
  - Gå framåt x steg
  - Vrid φ grader
  - Repetition

  Styrning från PC

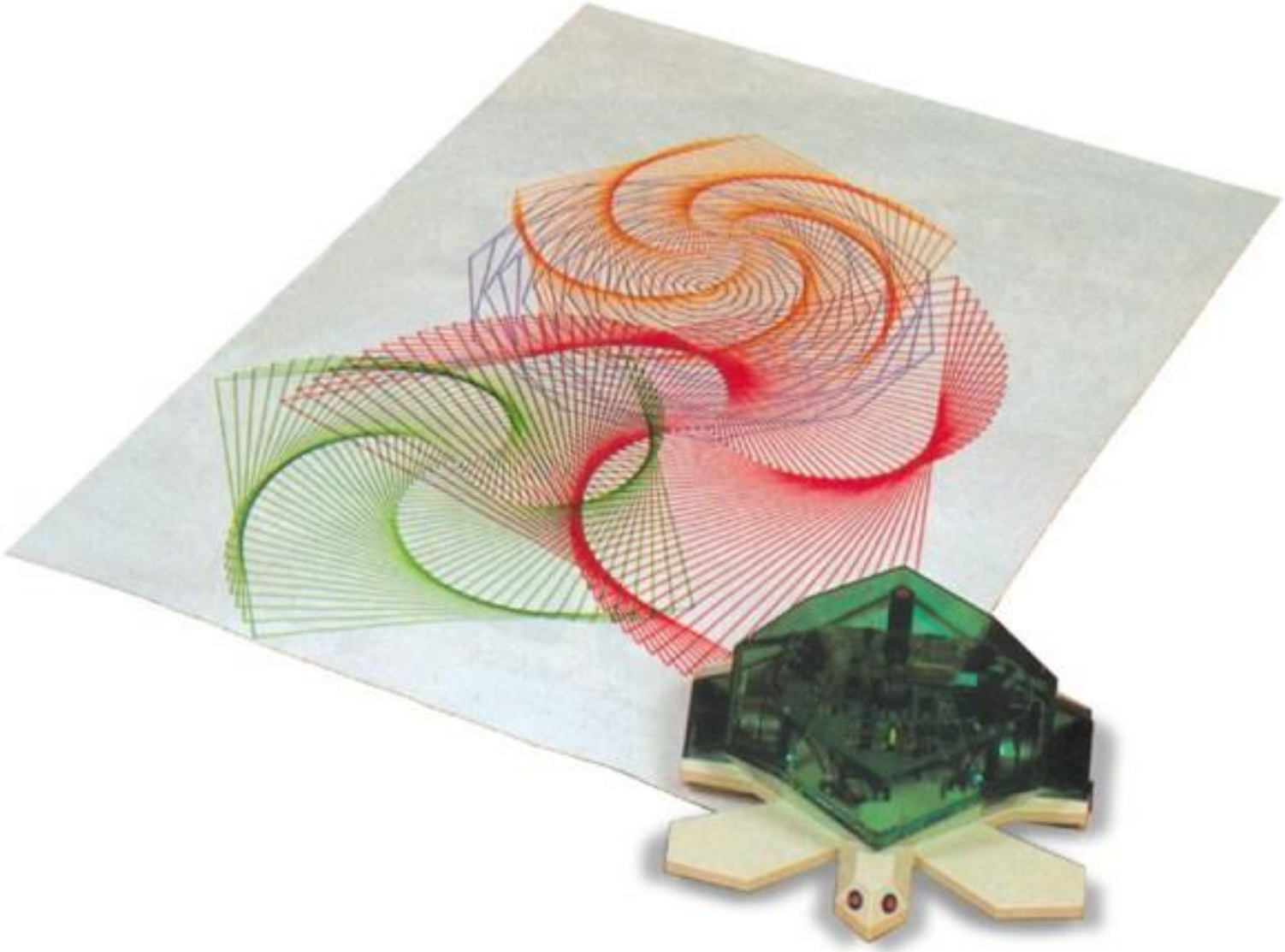
  Idéer till styrkommandon hittar du här !

  Du kan testa en webturtle här !



Varje student utvecklar eget program. Genomgång av planering av mjukvaruarkitektur i seminarie 2. Det är tillåtet att diskutera program-lösningar med andra men inte att redovisa likadana program.

Montering och lödning hjälps vi åt med. Fyra prototyper planeras.



Seminarie #1

- Diskutera i er grupp schemat för stegmotor-drivkretsen och hur den kopplas in till STM32F3 på discoverykortet. Bestäm även lämpliga komponentvärden i schemat.
- Gör detsamma för RC232-modulen.
- Föreslå också hur servo skall kopplas in till STM32. Servo skall användas för pennlyftet.