```lua
#!/usr/bin/env lua
-- vim: ts=2:sw=2:sts=2:et
local run

local function saturday(x) return math.floor(x)%7==6 end

-- Simple household diaper supply model
-- Buy weekly, use daily, dispose weekly (except when you forget)
local function diapers()
  return run({C={100,0,200}, -- clean diapers (stock)
              D={0,0,200},   -- dirty diapers (stock)
              q={0,0,100},   -- purchase rate (flow)
              r={8,0,20},    -- usage rate (flow)
              s={0,0,100}},  -- disposal rate (flow)
    function(dt,t,u,v)
      v.C = v.C + dt*(u.q-u.r)              -- clean += buy - use
      v.D = v.D + dt*(u.r-u.s)              -- dirty += use - dispose
      v.q = saturday(t) and 70 or 0         -- buy 70 on saturdays
      v.s = saturday(t) and u.D or 0        -- dispose all on saturdays
      if t==27 then v.s=0 end end) end      -- forgot to dispose on day 27

-- Brooks, F. (1975). The Mythical Man-Month. Addison-Wesley.
-- "Adding manpower to a late software project makes it later"
local function brooks()
  return run({D={20,0,100},    -- experienced developers (stock)
              N={0,0,100},     -- newbies (stock)
              W={0,0,1000},    -- work done (stock)
              R={1000,0,1000}}, -- work remaining (stock)
    function(dt,t,u,v)
      local comm = u.D*(u.D-1)/2*0.01       -- communication overhead (nÂ²)
      local train = u.N*0.2                 -- training overhead
      local prod = u.D*(1-comm-train)*10    -- actual productivity
      v.R = u.R - dt*math.max(0,prod)       -- remaining -= productivity
      v.W = u.W + dt*math.max(0,prod)       -- done += productivity
      v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0)  -- hire 10 at t=10
      v.D = u.D + dt*0.1*u.N end) end       -- newbies âM-^FM-^R experienced

-- Generic defect discovery model
-- Latent bugs discovered and fixed over time
local function bugs()
  return run({L={80,0,100},  -- latent bugs (stock)
              F={0,0,100},   -- found bugs (stock)
              X={0,0,100}},  -- fixed bugs (stock)
    function(dt,t,u,v)
      local find = u.L*0.15                 -- discovery rate
      local fix = u.F*0.3                   -- fix rate
      v.L = u.L - dt*find                   -- latent -= found
      v.F = u.F + dt*(find-fix)             -- found += discovered - fixed
      v.X = u.X + dt*fix end) end           -- fixed += fix rate

-- Cunningham, W. (1992). "The WyCash Portfolio Management System"
-- Technical debt slows velocity over time
local function debt()
  return run({F={0,0,100},   -- features (stock)
              D={0,0,100},   -- debt (stock)
              V={10,0,20}},  -- velocity (aux)
    function(dt,t,u,v)
      local add = u.V                       -- feature rate
      local accrue = add*0.1                -- debt per feature
      local repay = u.D*0.2                 -- debt repayment
      local slow = 1-u.D/100                -- debt slows velocity
      v.F = u.F + dt*add*slow               -- features += slowed rate
      v.D = u.D + dt*(accrue-repay)         -- debt += accrued - repaid
      v.V = u.V*slow end) end               -- velocity slows

-- Kermack & McKendrick (1927). doi:10.1098/rspa.1927.0118
-- SIR model adapted for defect propagation through code
local function sir()
  return run({S={90,0,100},  -- susceptible code (stock)
              I={10,0,100},  -- infected code (stock)
              R={0,0,100}},  -- removed/fixed (stock)
    function(dt,t,u,v)
      local infect = u.S*u.I*0.001          -- infection rate (SxI)
      local remove = u.I*0.15               -- fix rate
      v.S = u.S - dt*infect                 -- susceptible -= infected
      v.I = u.I + dt*(infect-remove)        -- infected += new - fixed
      v.R = u.R + dt*remove end) end        -- removed += fixed
```

```lua

-- Abdel-Hamid & Madnick (1991). Software Project Dynamics. Prentice-Hall
-- Development with testing and rework feedback
local function rework()
  return run({Req={100,0,100}, -- requirements (stock)
              Dev={0,0,100},   -- in development (stock)
              Test={0,0,100},  -- in testing (stock)
              Rew={0,0,100},   -- rework queue (stock)
              Done={0,0,100}}, -- completed (stock)
    function(dt,t,u,v)
      local code = u.Req*0.2                -- coding rate
      local test = u.Dev*0.3                -- testing rate
      local fail = u.Test*0.4               -- failure rate
      local pass = u.Test*0.6               -- pass rate
      local fix = u.Rew*0.5                 -- rework rate
      v.Req = u.Req - dt*code + dt*fix      -- req -= coded + reworked
      v.Dev = u.Dev + dt*code - dt*test     -- dev += coded - tested
      v.Test = u.Test + dt*test - dt*(fail+pass)  -- test += in - out
      v.Rew = u.Rew + dt*fail - dt*fix      -- rework += failed - fixed
      v.Done = u.Done + dt*pass end) end    -- done += passed

-- Generic learning/mentoring model
-- Juniors âM-^FM-^R trained âM-^FM-^R seniors âM-^FM-^R mentors
local function learn()
  return run({Jr={20,0,100},  -- juniors (stock)
              Tr={5,0,100},   -- in training (stock)
              Sr={5,0,100},   -- seniors (stock)
              Mn={0,0,100}},  -- mentoring (stock)
    function(dt,t,u,v)
      local train = u.Jr*0.1                -- training rate
      local promote = u.Tr*0.05             -- promotion rate
      local mentor = u.Sr*0.02              -- mentoring rate
      v.Jr = u.Jr - dt*train + dt*mentor    -- juniors -= training + new
      v.Tr = u.Tr + dt*train - dt*promote   -- training += in - promoted
      v.Sr = u.Sr + dt*promote - dt*mentor  -- seniors += promoted - mentors
      v.Mn = u.Mn + dt*mentor end) end      -- mentors += new

-- Brooks' Law extended with defect injection and escape
local function brooksq()
  return run({D={20,0,100},       -- experienced devs (stock)
              N={0,0,100},        -- newbies (stock)
              W={0,0,1000},       -- work done (stock)
              R={1000,0,1000},    -- remaining (stock)
              Defects={0,0,100}, -- defects (stock)
              Escapes={0,0,100}}, -- escaped defects (stock)
    function(dt,t,u,v)
      local comm = u.D*(u.D-1)/2*0.0001     -- communication overhead (scaled)
      local train = u.N*0.02                -- training overhead (scaled)
      local prod = u.D*(1-comm-train)*10    -- productivity
      local inject = prod*0.05              -- defects per work
      local escape = u.Defects*0.1          -- escape rate
      v.R = u.R - dt*math.max(0,prod)       -- remaining -= done
      v.W = u.W + dt*math.max(0,prod)       -- done += productivity
      v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0)  -- hire at t=10
      v.D = u.D + dt*0.1*u.N                -- newbies âM-^FM-^R experienced
      v.Defects = u.Defects + dt*inject - dt*escape  -- defects flow
      v.Escapes = u.Escapes + dt*escape end) end     -- escapes accumulate

-- Abdel-Hamid & Madnick (1991). Software Project Dynamics
-- Defect introduction, detection, residual, and operational discovery
local function defmap()
  return run({PC={20,0,100},    -- problem complexity (aux)
              DE={20,0,100},    -- design effort (aux)
              TE={2.5,0,10},    -- testing effort (aux)
              OU={35,0,100},    -- operational usage (aux)
              DI={3.43,0,100},  -- defects introduced (stock)
              DD={0,0,100},     -- defects detected (stock)
              RD={0,0,100},     -- residual defects (stock)
              OD={0,0,100}},    -- operational defects (stock)
    function(dt,t,u,v)
      local intro = u.PC*0.3 - u.DE*0.2     -- complexity adds, design removes
      local detect = u.TE*u.DI*0.4          -- testing detects
      local escape = u.DI*(1-u.TE*0.4)      -- undetected escape
      local oper = u.RD*u.OU*0.15           -- usage reveals residuals
      v.DI = u.DI + dt*intro                -- introduced += net
      v.DD = u.DD + dt*detect               -- detected += found
      v.RD = u.RD + dt*(escape-oper)        -- residual += escaped - found
      v.OD = u.OD + dt*oper                 -- operational += revealed
      v.PC,v.DE,v.TE,v.OU = u.PC,u.DE,u.TE,u.OU end) end  -- aux unchanged
```

```lua

-- Copy a table (shallow)
local function copy(t)
  local u={}; for k,v in pairs(t) do u[k]=v end; return u end

-- Run a compartmental model from time 0 to tmax
-- have: initial state {var={init,lo,hi},...}
-- step: function(dt,t,u,v) that updates v from u
function run(have,step,dt,tmax)
  dt,tmax = dt or 1, tmax or 30
  local t,u,keep = 0,{},{}
  for k,v in pairs(have) do u[k]=v[1] end  -- extract init values
  while t<tmax do
    local v=copy(u); step(dt,t,u,v)
    for k,h in pairs(have) do v[k]=math.max(h[2],math.min(h[3],v[k])) end  -- clamp
    keep[#keep+1]={t,v}; t,u = t+dt,v end
  return keep end

-- NUM: incremental stats
local function NUM() return {n=0, mu=0, m2=0, sd=0} end

local function add(i,z)
  i.n = i.n + 1; local d = z - i.mu
  i.mu = i.mu + d/i.n; i.m2 = i.m2 + d*(z - i.mu)
  i.sd = i.n<2 and 0 or math.sqrt(math.max(0,i.m2)/(i.n-1)); return z end

local function diff(num,a,b) return math.abs(a-b) > num.sd*0.35 end

local function show(keep)
  local cols={}
  for k,_ in pairs(keep[1][2]) do cols[#cols+1]=k end; table.sort(cols)
  local stats={}
  for _,col in ipairs(cols) do stats[col]=NUM() end
  for _,row in ipairs(keep) do
    for _,col in ipairs(cols) do add(stats[col],row[2][col]) end end
  io.write("t ")
  for _,col in ipairs(cols) do io.write(string.format("%6s ",col)) end; io.write("\n ")
  for _,col in ipairs(cols) do io.write(string.format("%6.1f ",stats[col].sd*0.35)) end;
 io.write("\n")
  local last={}
  for i,row in ipairs(keep) do
    io.write(string.format("%2d ",row[1]))
    for _,col in ipairs(cols) do
      if i==1 or diff(stats[col], last[col] or 0, row[2][col]) then
        io.write(string.format("%6.1f ",row[2][col])); last[col] = row[2][col]
      else io.write("   .") end end
    io.write("\n") end end

-- Main: run all models
for k,fun in pairs{diapers=diapers, brooks=brooks, bugs=bugs,
                   debt=debt, sir=sir, rework=rework,
                   learn=learn, brooksq=brooksq, defmap=defmap} do
  print("\n"..k..":"); show(fun()) end
```