

## compart.lua

Page 1/2

```

1  #!/usr/bin/env lua
2  -- vim: ts=2 sw=2: sts=2: et
3  local run
4
5  local function saturday(x) return math.floor(x)/7==6 end
6
7  -- Simple household diaper supply model
8  -- Buy weekly, use daily, dispose weekly (except when you forget)
9  local function diapers()
10     return run((C=(100,0,200), -- clean diapers (stock)
11                D=(0,0,200), -- dirty diapers (stock)
12                q=(0,0,100), -- purchase rate (flow)
13                r=(8,0,20), -- usage rate (flow)
14                s=(0,0,100)), -- disposal rate (flow)
15              function(dt,t,u,v)
16                  v.C = v.C + dt*(u.q-u.r) -- clean += buy - use
17                  v.D = v.D + dt*(u.r-u.s) -- dirty += use - dispose
18                  v.q = saturday(t) and 70 or 0 -- buy 70 on saturdays
19                  v.s = saturday(t) and u.D or 0 -- dispose all on saturdays
20                  if t==27 then v.s=0 end end end
21
22  -- Brooks, F. (1975). The Mythical Man-Month. Addison-Wesley.
23  -- Adding manpower to a late software project makes it later"
24  local function brooks()
25     return run((D=(20,0,100), -- experienced developers (stock)
26                N=(0,0,100), -- newbies (stock)
27                W=(0,0,1000), -- work done (stock)
28                R=(1000,0,1000)), -- work remaining (stock)
29              function(dt,t,u,v)
30                  local comm = u.D*(u.D-1)/2*0.01 -- communication overhead (n^2)
31                  local train = u.N*0.2 -- training overhead
32                  local prod = u.D*(1-comm-train)*10 -- actual productivity
33                  v.R = u.R - dt*math.max(0,prod) -- remaining -= productivity
34                  v.W = u.W + dt*math.max(0,prod) -- done += productivity
35                  v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0) -- hire 10 at t=10
36                  v.D = u.D + dt*0.1*u.N end end -- newbies 4M-7M-^R experienced
37
38  -- Generic defect discovery model
39  -- Latent bugs discovered and fixed over time
40  local function bugs()
41     return run((L=(80,0,100), -- latent bugs (stock)
42                F=(0,0,100), -- found bugs (stock)
43                X=(0,0,100)), -- fixed bugs (stock)
44              function(dt,t,u,v)
45                  local find = u.L*0.15 -- discovery rate
46                  local fix = u.F*0.3 -- fix rate
47                  v.L = u.L - dt*find -- latent -= found
48                  v.F = u.F + dt*(find-fix) -- found += discovered - fixed
49                  v.X = u.X + dt*fix end end -- fixed += fix rate
50
51  -- Cunningham, W. (1992). "The MyCash Portfolio Management System"
52  -- Technical debt slows velocity over time
53  local function debt()
54     return run((F=(0,0,100), -- features (stock)
55                D=(0,0,100), -- debt (stock)
56                V=(10,0,20)), -- velocity (aux)
57              function(dt,t,u,v)
58                  local add = u.V -- feature rate
59                  local accrue = add*0.1 -- debt per feature
60                  local repay = u.D*0.2 -- debt repayment
61                  local slow = 1-u.D/100 -- debt slows velocity
62                  v.F = u.F + dt*add*slow -- features += slowed rate
63                  v.D = u.D + dt*(accrue-repay) -- debt += accrued - repaid
64                  v.V = u.V*slow end end -- velocity slows
65
66  -- Kermack & McKendrick (1927). doi:10.1098/rspa.1927.0118
67  -- SIR model adapted for defect propagation through code
68  local function sir()
69     return run((S=(90,0,100), -- susceptible code (stock)
70                I=(10,0,100), -- infected code (stock)
71                R=(0,0,100)), -- removed/fixed (stock)
72              function(dt,t,u,v)
73                  local infect = u.S*u.I*0.001 -- infection rate (SxI)
74                  local remove = u.I*0.15 -- fix rate
75                  v.S = u.S - dt*infect -- susceptible -= infected
76                  v.I = u.I + dt*(infect-remove) -- infected += new - fixed
77                  v.R = u.R + dt*remove end end -- removed += fixed
78
79  -- Abdel-Hamid & Madnick (1991). Software Project Dynamics. Prentice-Hall
80  -- Development with testing and rework feedback
81  local function rework()
82     return run((Req=(100,0,100), -- requirements (stock)
83                Dev=(0,0,100), -- in development (stock)
84                Test=(0,0,100), -- in testing (stock)
85                Rework=(0,0,100), -- rework queue (stock)
86                Done=(0,0,100)), -- completed (stock)
87              function(dt,t,u,v)
88                  local code = u.Req*0.2 -- coding rate
89                  local test = u.Dev*0.3 -- testing rate
90                  local fail = u.Test*0.4 -- failure rate
91                  local pass = u.Test*0.6 -- pass rate
92                  local fix = u.Req*0.5 -- rework rate
93                  v.Req = u.Req - dt*code + dt*fix -- req -= coded + reworked
94                  v.Dev = u.Dev + dt*code - dt*test -- dev += coded - tested
95                  v.Test = u.Test + dt*test - dt*(fail+pass) -- test += in - out
96                  v.Rew = u.Rew + dt*fail - dt*fix -- rework += failed - fixed
97                  v.Done = u.Done + dt*pass end end -- done += passed
98
99  -- Generic learning/mentoring model
100  -- Juniors 4M-^FM-^R trained 4M-^FM-^R seniors 4M-^FM-^R mentors
101  local function learn()
102     return run((Jr=(20,0,100), -- juniors (stock)
103                Tr=(5,0,100), -- in training (stock)
104                Sr=(5,0,100), -- seniors (stock)
105                Mn=(0,0,100)), -- mentoring (stock)
106              function(dt,t,u,v)
107                  local train = u.Jr*0.1 -- training rate
108                  local promote = u.Tr*0.05 -- promotion rate
109                  local mentor = u.Sr*0.02 -- mentoring rate
110                  v.Jr = u.Jr - dt*train + dt*mentor -- juniors -= training + new
111                  v.Tr = u.Tr + dt*train - dt*promote -- training += in - promoted
112                  v.Sr = u.Sr + dt*promote - dt*mentor -- seniors += promoted - mentors
113                  v.Mn = u.Mn + dt*mentor end end -- mentors += new
114
115  -- Brooks' Law extended with defect injection and escape
116  local function brooksq()
117     return run((D=(20,0,100), -- experienced devs (stock)
118                N=(0,0,100), -- newbies (stock)
119                W=(0,0,1000), -- work done (stock)
120                R=(1000,0,1000), -- remaining (stock)

```

## compart.lua

Page 2/2

```

121     Defects={0,0,100}, -- defects (stock)
122     Escapes={0,0,100}}, -- escaped defects (stock)
123   function(dt,t,u,v)
124       local comm = u.D*(u.D-1)/2*0.0001 -- communication overhead (scaled)
125       local train = u.N*0.02 -- training overhead (scaled)
126       local prod = u.D*(1-comm-train)*10 -- productivity
127       local inject = prod*0.05 -- defects per work
128       local escape = u.Defects*0.1 -- escape rate
129       v.S = u.S - dt*math.max(0,prod) -- remaining -= done
130       v.W = u.W + dt*math.max(0,prod) -- done += productivity
131       v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0) -- hire at t=10
132       v.D = u.D + dt*1*u.N -- newbies 4M-7M-^R experienced
133       v.Defects = u.Defects + dt*inject - dt*escape -- defects flow
134       v.Escapes = u.Escapes + dt*escape end end -- escapes accumulate
135
136  -- Abdel-Hamid & Madnick (1991). Software Project Dynamics
137  -- Defect introduction, detection, residual, and operational discovery
138  local function defmap()
139     return run((PC=(20,0,100), -- problem complexity (aux)
140                DE=(20,0,100), -- design effort (aux)
141                TE=(2.5,0,10), -- testing effort (aux)
142                OU=(35,0,100), -- operational usage (aux)
143                DI=(3.43,0,100), -- defects introduced (stock)
144                DD=(0,0,100), -- defects detected (stock)
145                RD=(0,0,100), -- residual defects (stock)
146                OD=(0,0,100)), -- operational defects (stock)
147              function(dt,t,u,v)
148                  local intro = u.PC*0.3 - u.DE*0.2 -- complexity adds, design removes
149                  local detect = u.DE*u.DI*0.4 -- testing detects
150                  local escape = u.DI*(1-u.DE*0.4) -- undetected escape
151                  local oper = u.RD*u.OU*0.15 -- usage reveals residuals
152                  v.DI = u.DI + dt*intro -- introduced += net
153                  v.DD = u.DD + dt*detect -- detected += found
154                  v.RD = u.RD + dt*(escape-oper) -- residual += escaped - found
155                  v.OD = u.OD + dt*oper -- operational += revealed
156                  v.PC,v.DE,v.TE,v.OD = u.PC,u.DE,u.TE,u.OD end end -- aux unchanged
157
158  -- Copy a table (shallow)
159  local function copy(t)
160     local u={}; for k,v in pairs(t) do u[k]=v end; return u end
161
162  -- Run a compartmental model from time 0 to tmax
163  -- have: initial state (var={init,lo,hi},...)
164  -- step: function(dt,t,u,v) that updates v from u
165  local function run(have,step,dt,tmax)
166     dt,tmax = dt or 1, tmax or 30
167     local t,u,keep = 0,{},{}
168     for k,v in pairs(have) do u[k]=v[1] end -- extract init values
169     while t<tmax do
170         local v=copy(u); step(dt,t,u,v)
171         for k,h in pairs(have) do v[k]=math.max(h[2],math.min(h[3],v[k])) end -- clamp
172         keep[#keep+1]={t,v}; t,u = t+dt,v end
173     return keep end
174
175  -- NUM: incremental stats
176  local function NUM(t) return {n=0, mu=0, m2=0, sd=0} end
177
178  local function add(i,z)
179     i.n = i.n + 1; local d = z - i.mu
180     i.mu = i.mu + d/i.n; i.m2 = i.m2 + d*(z - i.mu)
181     i.sd = i.n<2 and 0 or math.sqrt(math.max(0,i.m2)/(i.n-1)); return z end
182
183  local function diff(num,a,b) return math.abs(a-b) > num.sd*0.35 end
184
185  local function show(keep)
186     local cols={}
187     for k,v in pairs(keep[1][2]) do cols[#cols+1]=k end; table.sort(cols)
188     local stats={}
189     for _,col in ipairs(cols) do stats[col]=NUM() end
190     for _,row in ipairs(keep) do
191         for _,col in ipairs(cols) do add(stats[col],row[2][col]) end end
192         io.write("\n")
193         for _,col in ipairs(cols) do io.write(string.format("%6s",col)) end; io.write("\n ")
194         for _,col in ipairs(cols) do io.write(string.format("%6.1f",stats[col].sd*0.35)) end;
195         io.write("\n")
196         local last={}
197         for i,row in ipairs(keep) do
198             io.write(string.format("%3d",row[1]))
199             for _,col in ipairs(cols) do
200                 if i==1 or diff(stats[col], last[col] or 0, row[2][col]) then
201                     io.write(string.format("%6.1f",row[2][col])); last[col] = row[2][col]
202                 else io.write(" ") end end
203             io.write("\n") end end
204
205  -- Main: run all models
206  for k,fun in pairs(diapers=diapers, brooks=brooks, bugs=bugs,
207                  debt=debt, sir=sir, rework=rework,
208                  learn=learn, brooksq=brooksq, defmap=defmap) do
209     print("\n"..k..":"); show(fun()) end

```