

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 forgot)
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=(0,0,20), -- usage rate (flow)
14             g=(0,0,100)), -- disposal rate (flow)
15
16   function(dt,t,u,v)
17     v.C = v.C + dt*(u.q-u.r) -- clean += buy - use
18     v.D = v.D + dt*(u.r-u.s) -- dirty += use - dispose
19     v.q = saturday(t) and 7 or 0 -- buy 70 on Saturdays
20     v.s = saturday(t) and u.D or 0 -- dispose all on Saturdays
21     if t==27 then v.s=0 end end -- forgot to dispose on day 27
22
23 -- Brooks, F. (1975). The Mythical Man-Month. Addison-Wesley.
24 -- "Adding manpower to a late software project makes it later"
25 local function brooks()
26   return run((D=(20,0,100), -- experienced developers (stock)
27             N=(0,0,100), -- newbies (stock)
28             W=(0,0,1000), -- work done (stock)
29             R=(1000,0,1000)), -- work remaining (stock)
30
31   local DE=u.D*(u.D-1)/2*0.01 -- communication overhead (nA)
32   local train = u.N*0.2 -- training overhead
33   local prod = u.*(1-comm-train)*10 -- actual productivity
34   v.R = u.R - dt*math.max(0,prod) -- remaining -= productivity
35   v.W = u.W + dt*math.max(0,prod) -- done += productivity
36   v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0) -- hire at t=10
37   v.D = u.D + dt*0.1*u.N end end -- newbies ^M-FM-R experienced
38
39 -- Generic defect discovery model
40 -- Latent bugs discovered and fixed over time
41 local function bugs()
42   return run((B=(80,0,100), -- latent bugs (stock)
43             F=(0,0,100), -- found bugs (stock)
44             X=(0,0,100)), -- fixed bugs (stock)
45
46   function(dt,t,u,v)
47     local find = u.L*0.15 -- discovery rate
48     local fix = u.F*0.3 -- fix rate
49     v.L = u.L - dt*find -- latent -= found
50     v.F = u.F + dt*(find-fix) -- found += discovered - fixed
51     v.X = u.X + dt*fix end end
52
53 -- Cunningham, W. (1992). "The MyCash Portfolio Management System"
54 -- Technical debt slows velocity over time
55 local function debt()
56   return run((F=(0,0,100), -- features (stock)
57             D=(0,0,100), -- debt (stock)
58             V=(10,0,20)), -- velocity (aux)
59
60   function(dt,t,u,v)
61     local add = u.V -- feature rate
62     local accrue = add*0.1 -- debt per feature
63     local repay = u.D*0.2 -- debt repayment
64     local slow = 1-u.D/100 -- debt slows velocity
65     v.F = u.F + dt*add*slow -- features += slowed rate
66     v.D = u.D + dt*(accrue-repay) -- debt += accrued - repaid
67     v.V = u.V*slow end end
68
69 -- Kermack & McKendrick (1927). doi:10.1098/rspa.1927.0118
70 -- SIR model adapted for defect propagation through code
71 local function sir()
72   return run((S=(90,0,100), -- susceptible code (stock)
73             I=(0,0,100), -- infected code (stock)
74             R=(0,0,100)), -- removed/fixed (stock)
75
76   function(dt,t,u,v)
77     local infect = u.S*u.I*0.001 -- infection rate (SXI)
78     local remove = u.I*0.15 -- fix rate
79     v.S = u.S - dt*infect -- susceptible -= infected
80     v.I = u.I + dt*(infect-remove) -- infected += new - fixed
81     v.R = u.R + dt*remove end end
82
83 -- Abdel-Hamid & Madnick (1991). Software Project Dynamics. Prentice-Hall
84 -- Development with testing and rework feedback
85 local function rework()
86   return run((Req=(100,0,100), -- requirements (stock)
87             Dev=(0,0,100), -- in development (stock)
88             Test=(0,0,100), -- in testing (stock)
89             Rew=(0,0,100), -- rework queue (stock)
90             Done=(0,0,100), -- completed (stock)
91
92   local code = u.Req*0.2 -- coding rate
93   local test = u.Dev*0.3 -- testing rate
94   local fail = u.Test*0.4 -- failure rate
95   local pass = u.Test*0.6 -- pass rate
96   local fix = u.Req*0.5 -- rework rate
97
98   v.Req = Req - dt*(done+dt*fix) -- req -= coded + reworked
99   v.Dev = u.Dev + dt*code - dt*test -- dev += coded - tested
100  v.Test = u.Test + dt*test - dt*(fail+pass) -- test += in - out
101  v.Rew = dt*fail - dt*fix -- rework += failed - fixed
102  v.Done = u.Done + dt*pass end end -- done += passed
103
104 -- Generic learning/mentoring model
105 -- Juniors ^M-FM-R trained ^M-FM-R seniors ^M-FM-R mentors
106 local function learn()
107   return run((Jr=(20,0,100), -- juniors (stock)
108             Tr=(5,0,100), -- in training (stock)
109             Sr=(5,0,100), -- seniors (stock)
110             Mn=(5,0,100), -- mentoring (stock)
111
112   function(dt,t,u,v)
113     local train = u.Jr*0.1 -- training rate
114     local promote = u.Tr*0.05 -- promotion rate
115     local mentor = u.Sr*0.02 -- mentoring rate
116
117     v.Jr = Jr - dt*train + dt*mentor -- juniors -= training + new
118     v.Tr = Tr - dt*train + dt*promote -- training += in - promoted
119     v.Sr = u.Sr + dt*promote - dt*mentor -- seniors += promoted - mentors
120     v.Mn = u.Mn + dt*mentor end end -- mentors += new
121
122 -- Brooks' Law extended with defect injection and escape
123 local function defmap()
124   return run((D=(20,0,100), -- experienced devs (stock)
125             N=(0,0,100), -- newbies (stock)
126             W=(0,0,1000), -- work done (stock)
127             R=(1000,0,1000), -- remaining (stock)

```

compart.lua

Page 2/2

```

128           Defects=(0,0,100), -- defects (stock)
129           Escapes=(0,0,100), -- escaped defects (stock)
130
131   function(dt,t,u,v)
132     local comm = u.D*(u.D-1)/2*0.0001 -- communication overhead (scaled)
133     local train = u.N*0.02 -- training overhead (scaled)
134     local prod = u.*(1-comm-train)*10 -- productivity
135     local inject = prod*0.05 -- defects per work
136     local Mn = u.Mn*0.1 -- mentors
137     v.R = u.R + dt*math.max(0,prod) -- remaining -= done
138     v.W = u.W + dt*math.max(0,prod) -- done += productivity
139     v.N = u.N - dt*0.1*u.N + (t==10 and 10 or 0) -- hire at t=10
140     v.D = u.D + dt*0.1*u.N -- newbie ^M-FM-R experienced
141     v.Defects = u.Defects + dt*inject - dt*escape -- defects flow
142     v.Escapes = u.Escapes + dt*escape end end -- escapes accumulate
143
144 -- Abdel-Hamid & Madnick (1991). Software Project Dynamics
145 -- Defect introduction, detection, residual, and operational discovery
146 local function defmap()
147   return run((P=(20,0,100), -- problem complexity (aux)
148             DE=(20,0,100), -- design effort (aux)
149             TE=(2,5,0,10), -- testing effort (aux)
150             OU=(35,0,100), -- operational usage (aux)
151             DI=(3,43,0,100), -- defects introduced (stock)
152             DD=(0,0,100), -- defects detected (stock)
153             RD=(0,0,100), -- residual defects (stock)
154             OD=(0,0,100), -- operational defects (stock)
155
156   function(dt,t,u,v)
157     local intro = u.PC*0.3 - u.DE*0.2 -- complexity adds, design removes
158     local detect = u.TE*u.DI*0.4 -- testing detects
159     local escape = u.TE*u.OD*0.4 -- undetected escapes
160     local usage = u.RD*u.OU*0.15 -- usage reveals residuals
161     v.DI = u.DI + dt*intro -- introduced += net
162     v.DD = u.DD + dt*detect -- detected += found
163     v.RD = u.RD + dt*(escape-oper) -- residual += escaped - found
164     v.OD = u.OD + dt*oper -- operational += revealed
165     v.PC = u.PC, u.DB, v.TE, v.OU = u.PC, u.DE, u.TE, u.OU end end -- aux unchanged
166
167 -- Copy a table (shallow)
168 local function copy(t)
169   local u={}; for k,v in pairs(t) do u[k]=v end; return u end
170
171 -- Run a compartmental model from time 0 to tmax
172 -- have: initial state (var=(init,lo,hi,...))
173 -- step: function(dt,t,u,v) that updates v from u
174 local function run(have,step,dt,tmax)
175   dt,tmax = dt or 1, tmax or 30
176   local u={}; for k,v in pairs(have) do u[k]=v end
177   while t<tmax do
178     local v=copy(u); step(dt,t,u,v)
179     for k,h in pairs(have) do v[k]=math.max(h[2],math.min(h[3],v[k])) end -- clamp
180     keep[#keep+1]=t,v; t,u = t+dt, v end
181   return v
182
183 -- NUM: incremental stats
184 local function NUM()
185   return (n=0, mu=0, m2=0, sd=0) end
186
187 local function add(i,z)
188   i.n = i.n + 1; local d = z - i.mu
189   i.mu = i.mu + d/i.n; i.m2 = i.m2 + d*(z - i.mu)
190   i.sd = i.n*2 and 0 or math.sqrt(math.max(0,i.m2)/(i.n-1)); return z end
191
192 local function diff(n,a,b) return math.abs(a-b) > num.sd*0.35 end
193
194 local function show(keep)
195   local cols={}
196   for k,_ in pairs(keep) do cols[#cols+1]=k end; table.sort(cols)
197   local stats={}
198   for _,col in ipairs(cols) do stats[col]=NUM() end
199   for _,row in ipairs(keep) do
200     for _,col in ipairs(row) do add(stats[col],row[2][col]) end end
201   io.write("\n")
202   for _,col in ipairs(cols) do io.write(string.format("%6s",col)) end; io.write("\n")
203   for _,col in ipairs(cols) do io.write(string.format("%6.1f",stats[col].sd*0.35)) end;
204   io.write("\n")
205   local row=1
206   for _,row in ipairs(keep) do
207     io.write(string.format("%2d",row[1]))
208     for _,col in ipairs(cols) do
209       if i.e1 or diff(stats[col],last[col] or 0, row[2][col]) then
210         io.write(string.format("%6.1f",row[2][col])); last[col] = row[2][col]
211       else io.write(" ") end end
212     io.write("\n") end end
213
214 -- Main: run all models
215 for k,fun in pairs(diapers=diapers, brooks=brooks, bugs=bugs,
216                     debt=debt, sir=sir, rework=rework,
217                     learn=learn, brooksg=brooksg, defmap=defmap) do
218   print(unpack(fun)); show(fun())
219

```