This file is part of cocomo. To know more, view the source code cocomo.py or read our home page.

Sample from the COCOMO Model

From the Boehm'00 book Software Cost Estimation with Cocomo II.

The `COCOMO2` code uses the following set of tunings that Boehm learned, sort of, from 161 projects from commercial, aerospace, government, and non-profit organizations– mostly from the period 1990 to 2000 (I saw "sort of" cause Boehm actually "fiddled" with these numbers, here and there, using his domain knowledge).

Overview

Q: What does this code do?
A: It extracts valid projects from ranges describing:

- Valid COCOMO ranges; a.k.a. "*Ranges(Base)*";
- The space of options within one project; a.k.a "*Ranges(Project)*";
- The suggested changes to that project; a.k.a. "*Ranges(Treatment)*";

The *intersection* of that that space is the *result* of changing a project.

Our goal is to use this tool to

- Assess planned changes;
- And to find better changes.

Example

E.g. Ranges(Base)

The space of legal values for a COCOMO project. That looks like this:

```
_ = None;  Coc2tunings = dict(
        #       vlow  low   nom   high  vhigh  xhigh
  Flex=[        5.07, 4.05, 3.04, 2.03, 1.01,    _],
  Pmat=[        7.80, 6.24, 4.68, 3.12, 1.56,    _],
  Prec=[        6.20, 4.96, 3.72, 2.48, 1.24,    _],
  Resl=[        7.07, 5.65, 4.24, 2.83, 1.41,    _],
  Team=[        5.48, 4.38, 3.29, 2.19, 1.01,    _],
```

```
  acap=[        1.42, 1.19, 1.00, 0.85, 0.71,    _],
  aexp=[        1.22, 1.10, 1.00, 0.88, 0.81,    _],
  cplx=[        0.73, 0.87, 1.00, 1.17, 1.34, 1.74],
  data=[           _, 0.90, 1.00, 1.14, 1.28,    _],
  docu=[        0.81, 0.91, 1.00, 1.11, 1.23,    _],
  ltex=[        1.20, 1.09, 1.00, 0.91, 0.84,    _],
  pcap=[        1.34, 1.15, 1.00, 0.88, 0.76,    _],
  pcon=[        1.29, 1.12, 1.00, 0.90, 0.81,    _],
  pexp=[        1.19, 1.09, 1.00, 0.91, 0.85,    _],
  pvol=[           _, 0.87, 1.00, 1.15, 1.30,    _],
  rely=[        0.82, 0.92, 1.00, 1.10, 1.26,    _],
  ruse=[           _, 0.95, 1.00, 1.07, 1.15, 1.24],
  sced=[        1.43, 1.14, 1.00, 1.00, 1.00,    _],
  site=[        1.22, 1.09, 1.00, 0.93, 0.86, 0.80],
  stor=[           _,    _, 1.00, 1.05, 1.17, 1.46],
  time=[           _,    _, 1.00, 1.11, 1.29, 1.63],
  tool=[        1.17, 1.09, 1.00, 0.90, 0.78,    _])
```

For this code:

- We use 1=v1ow, 2=low, 3=nom, 4=high, 5=vhigh, 6-xhigh.
- The first few variables decrease effort exponentially.
- To distinguish those *scale factors* from the rest of the code, we start them with an upper case letter.

E.g. Ranges(Project)

The space of legal values for a project.

In there any any uncertainties about that project then either:

- The project description does not mention that item;
- Or, that item is shown as a range of possible values.

```
@ok # all functions defining projects have the prefix "@ok"
def flight():
  "JPL Flight systems"
  return dict(
    kloc= xrange(7,418),
    Pmat = [2,3],          aexp = [2,3,4,5],
    cplx = [3,4,5,6],      data = [2,3],
    ltex = [1,2,3,4],      pcap = [3,4,5],
    pexp = [1,2,3,4],      rely = [3,4,5],
    stor = [3,4],          time = [3,4],
```

```
    acap = [3,4,5],
    sced = [3],
    tool = [2])
```

This is a decription of flight software from NASA's Jet Propulsion lab.

- Some things are known with certainity; e.g. this team makes very little use of *tools*.
  - Hence, *tool = [2]* has only one value
- Many things are uncertain so:
  - We do not mention "team cohesion" (a.k.a. *team*) so this can range very log to very high
  - We offer some things are ranges (e.g. *kloc* and "process maturity" *pmat*)

E.g. Ranges(treatment)

The planned change to the project.

For example, lets say someone decide to "treat" a project by improving personnel.

```
def improvePersonnel(): return dict(
  acap=[5],pcap=[5],pcon=[5], aexp=[5], pexp=[5], ltex=[5])
```

(Note that "improving personnel" is a sad euphism for sacking your current contractors and hiring new ones with maximum analyst and programming capability as well as programmer continuity, experience with analysis, platform and this development langauge.)

The COCOMO Equation.

Using the above, generate some estimates, measured in terms of *development months* where one month is 152 hours work by one developer (and includes development and management hours). For example, if *effort*=100, then according to COCOMO, five developers would finish the project in 20 *months*.

```
def COCOMO2(project, t=None,a=2.94, b=0.91):
  t=t or Coc2tunings # t = the big table of COCOMO tuning parameters
  sfs, ems, kloc = 0, 1, 10 # initializing some defaults
  for k,setting in project.items():
    if k == 'kloc':
      kloc = setting
```

```
    else:
      values = t[k]
      value  = values[setting - 1]
      if k[0].isupper: sfs += value
      else           : ems *= value
  return a * ems * kloc**(b + 0.01 * sfs)
```

LESSON 1: According to Boehm, development effort is exponential on lines of code But there is more to it that just size. Effort is changed linearly by a set of *effort multipliers* (`em`) and exponentially by some *scale factors* (`sf`).

```
scale          | Prec |  have we done this before?
factors        | Flex |  development flexibility
(exponentially | Resl |  any risk resolution activities?
 decrease      | Team |  team cohesion
 effort)       | Pmat |  process maturity
-------------------------------------------------------------
upper          | acap |  analyst capability
(linearly      | pcap |  programmer capability
 decrease      | pcon |  programmer continuity
 effort)       | aexp |  analyst experience
               | pexp |  programmer experience
               | ltex |  language and tool experience
               | tool |  use of tools
               | site |  multiple site development
               | sced |  length of schedule
-------------------------------------------------------------
lower          | rely |  required reliability
(linearly      | data |  secondary memory  storage requirements
 increase      | cplx |  program complexity
 effort)       | ruse |  software reuse
               | docu |  documentation requirements
               | time |  runtime pressure
               | stor |  main memory requirements
               | pvol |  platform volatility
```

LESSON 2 : The factors that effect delivery are not just what code is being developed. The above factors divide into:

- Product attributes: *what* is being developed (rely, data, clx, ruse, doco);
- Platform attributes: *where* is it being developed (time, stor, pvol);
- Personnel attributes: *who* is doing the work (acap, pcal, pcon, aexp, pexp, ltex);
- Project attributes: *how* is it being developed (tools, site, sced).
- And the *misc* scale factors: Prec, Flex, Resl, Team, Pmat.

Finding Ranges

We use the above to compute estimates for projects that have certain ranges. Recall from the above those ranges are the intersection of

- Valid COCOMO ranges (a.k.a. *Ranges(Base)*);
- The space of options within one project (a.k.a *Ranges(Project)*);
- The suggested changes to that project (a.k.a. Ranges(Treatment)_);

How do we specify all those ranges? Well. . .

Finding "Ranges(Base)"

To find *Ranges(Base)*, we ask the *Coc2tunings* table to report all the non-None indexes it supports.

```
def ranges(t=None):
  t = t or Coc2tunings
  out= {k:[n+1 for n,v in enumerate(lst) if v]
        for k,lst in t.items()}
  out["kloc"] = xrange(2,1001)
  return out
```

Two little details

- Note one cheat: I slipped in the range of value `kloc` values into `ranges` (2 to 1000).
- Using `ranges`, we can do a little defensive programming.
  - Suppose a function describes a project by return a dictionary whose keys are meant to be valid COCOMO variables and whose values are meant to be numbers for legal COCOMO ranges.
  - The following decorator calls that function at load time and compile time and checks that all its keys and values are valid.

```
def ok(f):
  all    = ranges()
  prefix = f.__name__
  for k,some in f().items():
    if k == 'nkloc': continue
    if not k in all:
      raise KeyError( '%s.%s' % (prefix,k))
    else:
      possible   = all[k]
```

```
      impossible = set(some) - set(possible)
      if impossible:
        raise IndexError( '%s.%s=%s' %
                          (prefix,k,impossible))
  return f
```

For an example of using this function, see below.

Defining "Ranges(Project)"

In practice, we rarely know all the exact COCOMO factors for any project with 100% certainty. So the real game with effort estimation is study estimates across a *space of possibilities*.

For example, after talking to some experts at NASA's Jet Propulsion Laboratory, here are some descriptors of various NASA projects. Some of these are point values (for example, for flight guidance systems, reliabilty must be as high as possible so we set it to its maximum value of 5). However, many other variables are really *ranges* of values representing the space of options within certain software being built at NASA.

Note the addition of *@ok* before each function. This means, at load time, we check that all the following variables and ranges are valid.

```
@ok
def flight():
  "JPL Flight systems"
  return dict(
    kloc= xrange(7,418),
    Pmat = [2,3],          aexp = [2,3,4,5],
    cplx = [3,4,5,6],      data = [2,3],
    ltex = [1,2,3,4],      pcap = [3,4,5],
    pexp = [1,2,3,4],      rely = [3,4,5],
    stor = [3,4],          time = [3,4],
    acap = [3,4,5],
    sced = [3],
    tool = [2])

@ok
def ground():
  "JPL ground systems"
  return dict(
    kloc=xrange(11,392),
    Pmat = [2,3],          acap = [3,4,5],
    aexp = [2,3,4,5],      cplx = [1,2,3,4],
    data = [2,3],          rely = [1,2,3,4],
```

```
    ltex = [1,2,3,4],     pcap = [3,4,5],
    pexp = [1,2,3,4],     time = [3,4],
    stor = [3,4],
    tool = [2],
    sced = [3])

@ok
def osp():
  "Orbital space plane. Flight guidance system."
  return dict(
    kloc= xrange(75,125),
    Flex = [2,3,4,5],     Pmat = [1,2,3,4],
    Prec = [1,2],         Resl = [1,2,3],
    Team = [2,3],         acap = [2,3],
    aexp = [2,3],         cplx = [5,6],
    docu = [2,3,4],       ltex = [2,3,4],
    pcon = [2,3],         tool = [2,3],
    ruse = [2,3,4],       sced = [1,2, 3],
    stor = [3,4,5],
    data = [3],
    pcap = [3],
    pexp = [3],
    pvol = [2],
    rely = [5],
    site = [3])

@ok
def osp2():
  """Osp, version 2. Note there are more restrictions
  here than in osp version1 (since as a project
  develops, more things are set in stone)."""
  return dict(
    kloc= xrange(75,125),
    docu = [3,4],         ltex = [2,5],
    sced = [2,3,4],       Pmat = [4,5],
    Prec = [3,4, 5],
    Resl = [4],           Team = [3],
    acap = [4],           aexp = [4],
    cplx = [4],           data = [4],
    Flex = [3],           pcap = [3],
    pcon = [3],           pexp = [4],
    pvol = [3],           rely = [5],
    ruse = [4],           site = [6],
    stor = [3],           time = [3],
    tool = [5])
```

```
@ok
def anything():
  "Anything goes."
  return ranges()
```

Defining "Ranges(Treatment)"

Lastly, we need to define what we are going to do to a project.

First, we define a little booking code that remembers all the treatments and, at load time, checks that the ranges are good.

```
def rx(f=None,all=[]):
  if not f: return all
  all += [f]
  return ok(f)
```

Ok, now that is done, here are the treatments. Note that they all start with *@rx*.

```
@rx
def doNothing(): return {}

@rx
def improvePersonnel(): return dict(
  acap=[5],pcap=[5],pcon=[5], aexp=[5], pexp=[5], ltex=[5])

@rx
def improveToolsTechniquesPlatform(): return dict(
  time=[3],stor=[3],pvol=[2],tool=[5], site=[6])

@rx
def improvePrecendentnessDevelopmentFlexibility(): return dict(
  Prec=[5],Flex=[5])

@rx
def increaseArchitecturalAnalysisRiskResolution(): return dict(
  Resl=[5])

@rx
def relaxSchedule(): return dict(
  sced = [5])

@rx
def improveProcessMaturity(): return dict(
  Pmat = [5])
```

```
@rx
def reduceFunctionality(): return dict(
  data = [2], nkloc=[0.5]) # nloc is a special symbol. Used to change kloc.

@rx
def improveTeam(): return dict(
  Team = [5])

@rx
def reduceQuality():  return dict(
  rely = [1], docu=[1], time = [3], cplx = [1])
```

Under the Hood: Complete-ing the Ranges.

Now that we have defined *Ranges(Base), Ranges(Project), and Ranges(Treatment)*, we need some tool to generate the ranges of the currnet project, given some treatment. In the following code:

- `ranges()` looks up the ranges for `all` COCOMO values; i.e. the *Ranges(Base)*
- `project()` accesses *Ranges(Project)*. For each of those ranges, we override *Ranges(Base)*.
- Then we impose *Ranges(Treatment)* to generate a list of valid ranges consistent with _Ranges(*)_
- The guesses from the project are then added to `ask` which pulls on value for each attribute.

```
def ask(x):
  return random.choice(list(x))
```

And here's the code:

```
def complete(project, rx=None, allRanges=ranges()):
  rx = rx or {}
  p = project()
  for k,default in allRanges.items():
    if not k in p:
      p[k] = default
  for k,rx1 in rx.items():
    if k == 'nkloc':
        p['kloc'] = [ ask(p['kloc']) * rx1[0] ]
    else:
      overlap = list(set(p[k]) & set(rx1))
```

```
      if overlap:
        p[k] = overlap
  return  {k: ask(x) for k,x in p.items()}
```

For example, here some code to generate projects that are consistent with what we know about `flight` projects. Note that we call it three times and get three different project:

```
>>> for _ in range(3):
      print("\n",complete(flight))

==>
 {'sced': 3, 'cplx': 5, 'site': 2, 'Prec': 3, 'Pmat': 3,
  'acap': 3, 'Flex': 3, 'rely': 5, 'data': 2, 'tool': 2,
  'pexp': 3, 'pcon': 1, 'aexp': 4, 'stor': 4, 'docu': 5,
  'Team': 5, 'pcap': 5, 'kloc': 41, 'ltex': 1, 'ruse': 6,
  'Resl': 2, 'time': 4, 'pvol': 3}

 {'sced': 3, 'cplx': 6, 'site': 5, 'Prec': 4, 'Pmat': 2,
  'acap': 3, 'Flex': 2, 'rely': 5, 'data': 3, 'tool': 2,
  'pexp': 1, 'pcon': 2, 'aexp': 4, 'stor': 3, 'docu': 1,
  'Team': 3, 'pcap': 5, 'kloc': 63, 'ltex': 2, 'ruse': 3,
  'Resl': 4, 'time': 3, 'pvol': 3}

 {'sced': 3, 'cplx': 5, 'site': 5, 'Prec': 4, 'Pmat': 2,
  'acap': 5, 'Flex': 5, 'rely': 3, 'data': 3, 'tool': 2,
   'pexp': 3, 'pcon': 5, 'aexp': 4, 'stor': 4, 'docu': 1,
   'Team': 3, 'pcap': 4, 'kloc': 394, 'ltex': 2, 'ruse': 2,
   'Resl': 1, 'time': 3, 'pvol': 5}
```

Using This Code

Finally, we can generate a range of estiamtes out of this code.

The following code builds *sample* number of projects and scores each one with *COCOMO2* (later, we will score thse projects in other ways). The results are pretty-printed using a utility called *xtiles*.

```
def sample(samples=1000,
           projects=[anything],
           treatments=[doNothing],
           score=COCOMO2):
  samples = 1000
  results = []
  for project in projects:
```

```
    for treatment in treatments:
      what     = (project.__name__,treatment.__name__)
      result   = [score(complete(project,treatment()))
                  for _ in xrange(samples)]
      results += [[what] + result]
  xtiles(results,width=30,show="%7.1f")
```

For example:

```
>>> sample(projects=[flight,anything])
```

```
rank | rx                          | median  |
==== | ==                          | ======= |
1    | ('flight', 'doNothing')     |  2696.9 | ( -*-           |              ),   35.1,  1153
2    | ('anything', 'doNothing') |  8254.4 | (   ----*---    |              ),    6.9,  3610
```

Here's code to try all our treatments on all our projects:

```
PROJECTS  = [flight,ground,osp,osp2,anything]
TREATMENTS= [doNothing, improvePersonnel, improveToolsTechniquesPlatform,
             improvePrecendentnessDevelopmentFlexibility,
             increaseArchitecturalAnalysisRiskResolution, relaxSchedule,
             improveProcessMaturity, reduceFunctionality]


def _effortsTreated():
  for project in PROJECTS:
    print("\n#### ",project.__name__," ","#"*50,"\n")
    sample(projects=[project],treatments=TREATMENTS)
```

If executed, this generates the following:

FLIGHT

```
rank | rx                                                  | median  |
==== | ==                                                  | ======= |
1    | ('flight', 'reduceFunctionality')                   |  1194.6 | ( --*-
2    | ('flight', 'improvePrecendentnessDevelopmentFlexibility') |  2222.4 | (   ----*---
2    | ('flight', 'increaseArchitecturalAnalysisRiskResolution') |  2280.9 | (   ---*---
2    | ('flight', 'improveToolsTechniquesPlatform')        |  2628.5 | (   ----*----
2    | ('flight', 'relaxSchedule')                         |  2855.0 | (   ----*----
2    | ('flight', 'improvePersonnel')                      |  2883.2 | (   -----*---
2    | ('flight', 'doNothing')                             |  2894.9 | (   -----*---
2    | ('flight', 'improveProcessMaturity')                |  3035.1 | (   -----*----
```

GROUND

```
rank | rx                                                    | median  |
==== | ==                                                    | ======= |
1    | ('ground', 'reduceFunctionality')                     |  1070.3 | ( --*-
2    | ('ground', 'improvePrecendentnessDevelopmentFlexibility') |  2039.5 | (   ----*---
2    | ('ground', 'increaseArchitecturalAnalysisRiskResolution') |  2352.2 | (   ----*--
2    | ('ground', 'doNothing')                               |  2370.8 | (   ----*----
2    | ('ground', 'relaxSchedule')                           |  2415.5 | (   ----*---
2    | ('ground', 'improvePersonnel')                        |  2439.6 | (   ----*----
2    | ('ground', 'improveToolsTechniquesPlatform')          |  2633.6 | (    ----*---
2    | ('ground', 'improveProcessMaturity')                  |  2671.1 | (   -----*---
```

OSP

```
rank | rx                                                    | median  |
==== | ==                                                    | ======= |
1    | ('osp', 'reduceFunctionality')                        |   518.0 | (  --*          |
2    | ('osp', 'improvePrecendentnessDevelopmentFlexibility') |  1200.2 | (          ---*|-
3    | ('osp', 'improvePersonnel')                           |  1291.1 | (           ---*-
3    | ('osp', 'increaseArchitecturalAnalysisRiskResolution') |  1291.8 | (           ---*-
3    | ('osp', 'relaxSchedule')                              |  1293.7 | (           ---*-
3    | ('osp', 'improveToolsTechniquesPlatform')             |  1296.8 | (           ---*-
3    | ('osp', 'improveProcessMaturity')                     |  1297.6 | (           ---*-
3    | ('osp', 'doNothing')                                  |  1299.8 | (           ---*-
```

OSP2

```
rank | rx                                                    | median  |
==== | ==                                                    | ======= |
1    | ('osp2', 'reduceFunctionality')                       |   342.4 | (  --*          |
2    | ('osp2', 'improveProcessMaturity')                    |   764.0 | (             -|
2    | ('osp2', 'improvePrecendentnessDevelopmentFlexibility') |   769.0 | (             -|
2    | ('osp2', 'increaseArchitecturalAnalysisRiskResolution') |   789.6 | (              |
2    | ('osp2', 'improvePersonnel')                          |   797.6 | (             -|
2    | ('osp2', 'doNothing')                                 |   797.9 | (             -|
2    | ('osp2', 'improveToolsTechniquesPlatform')            |   805.8 | (             -|
2    | ('osp2', 'relaxSchedule')                             |   811.8 | (              |
```

ANYTHING (all COOCMO)

```
rank | rx                                                    | median  |
==== | ==                                                    | ======= |
1    | ('anything', 'reduceFunctionality')                   |  3230.2 | ( -*-
```

```
2    | ('anything', 'improvePrecendentnessDevelopmentFlexibility') |  6475.5 | (  ---*--
2    | ('anything', 'increaseArchitecturalAnalysisRiskResolution') |  6490.5 | (  --*---
2    | ('anything', 'improveProcessMaturity')                      |  6845.9 | (  ---*--
2    | ('anything', 'improveToolsTechniquesPlatform')              |  7424.2 | (  ---*--
2    | ('anything', 'improvePersonnel')                            |  7470.7 | (  ---*---
2    | ('anything', 'relaxSchedule')                               |  7828.1 | (   --*---
2    | ('anything', 'doNothing')                                   |  8212.3 | (  ----*---
```