OMO: Software cost estimation

Tim Menzies

Lane Department of Computer Science, West Virginia University, PO Box 6109, Morgantown, WV, 26506-6109, USA;
`http://tim.menzies.us`; `tim@menzies.us`

Wp ref: `~menzies/src/pl/prod/omo.pl`, January 30, 2003.

Abstract COCOMO is a software effort estimation tool. OMO is COCOMO written in SWI-Prolog [5]

Contents

List of Figures

1 Warning!!!

Contains deliberate error. Output should be:

```
COCOMO.ga says 1223.53 months (total);
28 staff over 45 months
```

But show how that is all broken.

```
COCOMO.ga says 134588.0 months (total);
547 staff over 247 months
```

What is wrong?

2 What is COCOMO?

The COCOMO project aims at developing an open-source, public-domain software effort estimation model. The project has collected information on 161 projects from commercial, aerospace, government, and non-profit organizations [1, 4]. As of 1998, the projects represented in the database were of size 20 to 2000 KSLOC (thousands of lines of code) and took between 100 to 10000 person months to build.

COCOMO measures effort in calendar months where one month is 152 hours (and includes development and management hours). The core intuition behind COCOMO-based estimation is that as systems grow in size, the effort required to create them grows exponentially, i.e. $effort \propto KSLOC^x$. More precisely:

$$months = a * \left(KSLOC^{\left(0.91 + \sum_{i=1}^{5} SF_i\right)}\right) * \left(\prod_{j=1}^{17} EM_j\right)$$

where a is a domain-specific parameter, and KSLOC is estimated directly or computed from a function point analysis. SF_i are the scale factors (e.g. factors such as "have we built this kind of system before?") and EM_j are the cost drivers (e.g. required level of reliability). Figure 1 lists the scale drivers and effort multipliers.

Software effort-estimation models like COCOMO-II should be tuned to their local domain. Off-the-shelf "untuned" models have been up to 600% inaccurate in their estimates, e.g. [3, p165] and [2]. However, tuned models can be far more accurate. For example, [1] reports a study with a bayesian

Type	Acronym	Definition	Low-end	Medium	High-end
EM	acap	analyst capability	worst 15%	55%	best 10%
EM	aexp	applications experience	2 months	1 year	6 years
SF	arch	architecture or risk resolution	few interfaces defined or few risk eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
EM	cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
EM	data	database size (DB bytes/ Program SLOC)	10	100	1000
EM	docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
SF	flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
EM	ltex	language and tool-set experience	2 months	1 year	6 years
EM	pcap	programmer capability	worst 15%	55%	best 10%
EM	pcon	personnel continuity (% turnover per year)	48%	12%	3%
EM	pexp	platform experience	2 months	1 year	6 years
SF	pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
SF	prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
EM	pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
EM	rely	required reliability	errors mean slight inconvenience	errors are easily recoverable	errors can risk human life
EM	ruse	required reuse	none	across program	across multiple product lines
EM	sced	dictacted development schedule	deadlines moved closer to 75% of the original estimate	no change	deadlines moved back to 160% of the original estimate
EM	site	multi-site development	some contact: phone, mail	some email	interactive multi-media
EM	stor	main storage constraints (% of available RAM)	N/A	50%	95%
SF	team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
EM	time	execution time constraints (% of available CPU)	N/A	50%	95%
EM	tool	use of software tools	edit,code,debug		well intergrated with lifecycle

Fig. 1 Parameters of the COCOMO-II effort risk model; adapted from `http://sunset.usc.edu/COCOMOII/expert_cocomo/drivers.html`. "`Stor`" and "`time`" score "N/A"" for low-end values since they have no low-end defined in COCOMO-II. "SF" denotes "scale factors" and "EM" denotes "effort multipliers".

tuning algorithm using the COCOMO project database. After bayesian tuning, a cross-validation study showed that COCOMO-II model produced estimates that are within 30% of the actuals, 69% of the time.

Figure 2 shows the sizes of various COCOMO tuning parameters. Notice the linear fits of the top two tunings: these were generated via linear regression and hence are straight lines. The bottom row shows tunings generated from a genetic algorithm (GA): such GAs were designed to handle non-linear situations so their curve fits can be all over the place.

The intuition to be gained from Figure 2 is that some COCOMO parameters are more influential than others. Some are weakly correlated to increasing effort (column 1); some are weakly correlated to decreasing effort (column 2); and some are strongly correlated to decreasing effort (column 3). This will be useful later when we write search engines to control COCOMO. A core heuristic will be "change the influential parameters first".

The last column of Figure 2 relate to the effort multipliers. While shown here as linear, their influence can be even greater than that since they are used up in an exponential equation.

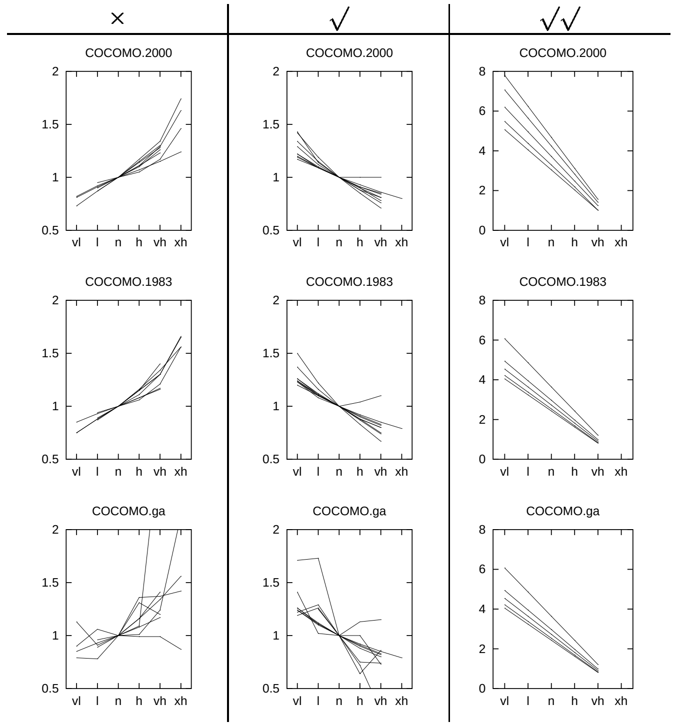

Fig. 2 Influence of different COCOMO parameters

3 Installation

```
:- load_files([lib % grab standard stuff
              ,cfg % options controller
              ,gpl0, gpl1 % GPL-2 license stuff
              ,omo0 % pre-load actions
          ,omolib % local libraries
              ,omo1 % predicates
              ,omo2 % start-up commands
          ,omokb1 % example project file
          ,ufp2sloc % function points per LOC database
              ],[silent(yes),if(changed)]).
```

4 Pre-load actions

4.1 Hooks

Fast assertions of named variables.

```
term_expansion((X;Y :- Z),Out) :-
    multis(((X;Y) :- Z),Out).
```

Instantiate named fields

```
term_expansion(Functor is Fields,Out) :-
    fields(Fields,Functor,Out).
```

5 Main System

5.1 Main driver

```
estimate :-
    cocomo(Coc),
    estimate(Pm,Staff,Months),
    format('COCOMO.~p says ~p months (total);',[Coc,Pm]),
    format('~p staff over ~p months\n', [Staff,Months]).

estimate(Pm,Staff,Months) :-
    tdev(Tdev),
    pm(Pm0),
    Pm is Pm0,
    Staff is ceiling(Pm/Tdev),
    Months is ceiling(Tdev),
    !.
```

5.2 Equations

5.2.1 Sizing equations

```
size((1 + (R/100)) *(N + E)) :-
    revl(R), newKsloc(N), equivalentKsloc(E).

equivalentKsloc(Ak*Aam*(1-(At/100))) :-
    adaptedKsloc(Ak), at(At), aam(Aam).

aam(Am) :- aaf(Af), compare(C,Af,50), aam1(C,Af,Am).

aam1(=,Af, X) :- aam1(<,Af,X).
aam1(>,Af, (Aa+Af+(Su*U))/100) :- aa(Aa),su(Su),unfm(U).
aam1(<,Af,((Aa+Af*(1+(0.02*Su*U)))/100)) :-
    aa(Aa), su(Su), unfm(U).

aaf(0.4*Dm+0.3*Cm+0.3*Im) :- dm(Dm), cm(Cm), im(Im).
```

5.2.2 Schedule Equations

```
tdev((C*(P^F))*SP/100) :-
    c(C), pmNs(P), f(F), scedPercent(SP).

f( D + 0.2*(E-B))  :-
    d(D),e(E), b(B).
```

5.2.3 Effort Equations

```
hmmm... sced value never used
pm(Pm0*Em17+Pa) :-
    pmNs(Pm0), w(sced,Em17), pmAuto(Pa).

pmNs(A*S*(S^E)*Em1 *Em2 *Em3 *Em4 *Em5 *Em6 *Em7*Em8*Em9*
            Em10*Em11*Em12*Em13*Em14*Em15*Em16) :-
    a(A), size(S), e(E), w(rely,Em1), w(data,Em2),
    w(cplx,Em3), w(ruse,Em4), w(docu,Em5), w(time,Em6),
    w(stor,Em7), w(pvol,Em8), w(acap,Em9), w(pcap,Em10),
    w(pcon,Em11),w(aexp,Em12),w(pexp,Em13),
    w(ltex,Em14),w(tool,Em15),w(site,Em16).

e(B + 0.01*(Sf1+Sf2+Sf3+Sf4+Sf5)) :-
    b(B),
    w(prec,Sf1), w(flex,Sf2),w(arch,Sf3),
    w(team,Sf4), w(pmat,Sf5).

pmAuto((Ak*(At/100))/Ap) :-
    adaptedKsloc(Ak), at(At), atKprod(Ap).
```

5.3 Tunings

5.3.1 Constants

```
a(2.5)  :- cocomo(1983).
a(2.94) :- cocomo(2000).
a(2.94) :- cocomo(ga).

b(0.91) :- cocomo(2000).
b(1.01) :- cocomo(1983).
b(1.01) :- cocomo(ga).

c(3.0)  :- cocomo(1983).
c(3.67) :- cocomo(2000).
c(3.67) :- cocomo(ga).

d(0.28) :- cocomo(2000).
d(0.33) :- cocomo(1983).
d(0.33) :- cocomo(ga).
```

5.3.2 Post-architecture scale factors The COCOMO 2000 scale factors learnt via bayesian tuning.

```
postArch(2000,scaleFactors) =
           [ xl, vl,   l,   n,   h,  vh,  xh]+
    [[prec,  _,6.20,4.96,3.72,2.48,1.24,   _]
    ,[flex,  _,5.07,4.05,3.04,2.03,1.01,   _]
    ,[arch,  _,7.07,5.65,4.24,2.83,1.41,   _]
    ,[team,  _,5.48,4.38,3.29,2.19,1.01,   _]
    ,[pmat,  _,7.80,6.24,4.68,3.12,1.56,   _]
    ].
```

The original scale factors.

```
postArch(1983,scaleFactors) =
           [ xl,  vl,   l,   n,   h,  vh, xh]+
    [[prec,   _,4.05,3.24,2.43,1.62,0.81,  _]
    ,[flex,   _,6.07,4.86,3.64,2.43,1.21,  _]
    ,[arch,   _,4.22,3.38,2.53,1.69,0.84,  _]
    ,[team,   _,4.94,3.95,2.97,1.98,0.99,  _]
    ,[pmat,   _,4.54,3.64,2.73,1.82,0.91,  _]
    ].
```

Some scale factors learnt via some genetic algorithms.

```
postArch(ga,scaleFactors) =
            [ xl,  vl,   l,   n,   h,  vh,  xh]+
    [[prec,   _,4.05,3.24,2.43,1.62,0.81,    _]
    ,[flex,    _,6.07,4.86,3.64,2.43,1.21,   _]
    ,[arch,    _,4.22,3.38,2.53,1.69,0.84,   _]
    ,[team,    _,4.94,3.95,2.97,1.98,0.99,   _]
    ,[pmat,    _,4.54,3.64,2.73,1.82,0.91,   _]
    ].
```

5.3.3 Post-architecture effort multipliers: The COCOMO 2000 effort multipliers learnt via bayesian tuning.

```
postArch(2000,effortMultiplers) =
           [xl,   vl,   l,   n,  h,  vh,  xh]+
    [[rely,  _,0.82,0.92,1.00,1.10,1.26,   _]
    ,[data,  _,_   ,0.90,1.00,1.14,1.28,   _]
    ,[cplx,  _,0.73,0.87,1.00,1.17,1.34,1.74]
    ,[ruse,  _,_   ,0.95,1.00,1.07,1.15,1.24]
    ,[docu,  _,0.81,0.91,1.00,1.11,1.23,   _]
    ,[time,  _,   _,   _,1.00,1.11,1.29,1.63]
    ,[stor,  _,   _,   _,1.00,1.05,1.17,1.46]
    ,[pvol,  _,   _,0.87,1.00,1.15,1.30,   _]
    ,[acap,  _,1.42,1.19,1.00,0.85,0.71 ,  _]
    ,[pcap,  _,1.34,1.15,1.00,0.88,0.76,   _]
    ,[pcon,  _,1.29,1.12,1.00,0.90,0.81,   _]
    ,[aexp,  _,1.22,1.10,1.00,0.88,0.81,   _]
    ,[pexp,  _,1.19,1.09,1.00,0.91,0.85,   _]
    ,[ltex,  _,1.20,1.09,1.00,0.91,0.84,   _]
    ,[tool,  _,1.17,1.09,1.00,0.90,0.78,   _]
    ,[site,  _,1.22,1.09,1.00,0.93,0.86,0.80]
    ,[sced,  _,1.43,1.14,1.00,1.00,1.00,   _]
    ].
```

The original effort multipliers.

```
postArch(1983,effortMultiplers) =
           [ xl,  vl,   l,   n,   h,  vh,  xh]+
    [[rely,   _,0.75,0.88,1.00,1.15,1.40,   _]
    ,[data,   _,   _,0.94,1.00,1.08,1.16,   _]
    ,[cplx,   _,0.75,0.88,1.00,1.15,1.30,1.65]
    ,[ruse,   _,   _,0.89,1.00,1.16,1.34,1.56]
    ,[docu,   _,0.85,0.93,1.00,1.08,1.17,   _]
    ,[time,   _,   _,   _,1.00,1.11,1.30,1.66]
    ,[stor,   _,   _,   _,1.00,1.06,1.21,1.56]
    ,[pvol,   _,   _,0.87,1.00,1.15,1.30,   _]
    ,[acap,   _,1.50,1.22,1.00,0.83,0.67,   _]
    ,[pcap,   _,1.37,1.16,1.00,0.87,0.74,   _]
    ,[pcon,   _,1.26,1.11,1.00,0.91,0.83,   _]
    ,[aexp,   _,1.23,1.10,1.00,0.88,0.80,   _]
    ,[pexp,   _,1.26,1.12,1.00,0.88,0.80,   _]
    ,[ltex,   _,1.24,1.11,1.00,0.90,0.82,   _]
    ,[tool,   _,1.20,1.10,1.00,0.88,0.75,   _]
    ,[site,   _,1.24,1.10,1.00,0.92,0.85,0.79]
    ,[sced,   _,1.23,1.08,1.00,1.04,1.10,   _]
    ].
```

Some effort multipliers learnt via some genetic algorithms.

```
postArch(ga,effortMultiplers) =
            [ xl,  vl,   l,   n,   h,  vh,  xh]+
    [[rely,    _,0.79,0.78,1.00,1.16,1.41,   _]
    ,[data,    _,   _,0.96,1.00,1.31,1.20,   _]
    ,[cplx,    _,0.90,1.06,1.00,0.99,0.99,0.87]
    ,[ruse,    _,   _,0.89,1.00,1.16,1.34,1.56]
    ,[docu,    _,0.85,0.93,1.00,1.08,1.17,   _]
    ,[time,    _,   _,   _,1.00,1.01,1.24,2.13]
    ,[stor,    _,   _,   _,1.00,1.36,1.37,1.42]
    ,[pvol,    _,   _,1.25,1.00,1.13,1.15,   _]
    ,[acap,    _,1.19,1.26,1.00,1.00,0.73,   _]
    ,[pcap,    _,1.71,1.73,1.00,0.75,0.74,   _]
    ,[pcon,    _,1.26,1.11,1.00,0.91,0.83,   _]
    ,[aexp,    _,1.41,1.02,1.00,0.64,0.86,   _]
    ,[pexp,    _,1.26,1.12,1.00,0.88,0.80,   _]
    ,[ltex,    _,1.24,1.11,1.00,0.90,0.82,   _]
    ,[tool,    _,1.13,0.91,1.00,1.09,2.86,   _]
    ,[site,    _,1.24,1.10,1.00,0.92,0.85,0.79]
    ,[sced,    _,1.22,1.29,1.00,0.72,0.29,   _]
    ].
```

5.4 Data dictionary

5.4.1 General

```
languageP(X) :- upf2sloc(X,_).

sym(X) :- rsym(X).

onezeroP(X) :- rin(0,1,0.2,X), number(X).

percentP(X) :- rin(0,100,1,X),integer(X).

posint(X)   :- rin(0,65536,X),integer(X).
posnum(X)   :- rin(0,inf,X),number(X).

num10(X) :- rin(0,10,X), number(X).

cocomoP(2000).
cocomoP(1983).
cocomoP(ga).

vlvh(n). vlvh(l). vlvh(h). vlvh(vl). vlvh(vh).

lvh(n). lvh(l). lvh(h). lvh(vh).

vlxh(n).  vlxh(l).  vlxh(h).
vlxh(vl). vlxh(vh). vlxh(xh).

lxh(n). lxh(l). lxh(h). lxh(vh). lxh(xh).

nxh(n). nxh(h). nxh(vh). nxh(xh).
```

5.4.2 "project"

```
(cocomo(Coc); label(L); language(Lan)
;revl(R); newKsloc(K)
;adaptedKsloc(A) ;cm(C); dm(D); im(I) ;aa(Aa) ;unfm(U)
;su(Su) ;at(At) ;atKprod(Atp) ;scedPercent(Sc)
) :-
    project(Coc,L,Lan,R,K,A,C,D,I,Aa,U,Su,At,Atp,Sc),

    cocomoP(Coc),
    sym(L), languageP(Lan), percentP(R), percentP(K),
    posint(A), percentP(C), percentP(I), percentP(Aa),
    onezeroP(U), percentP(Su) ,percentP(At),
    posnum(Atp) ,posint(Sc),!.
```

5.4.3 "scores"

```
(s(prec,Prec) ;s(flex,Flex) ;s(arch,Arch)
;s(team,Team) ;s(pmat,Pmat) ;s(rely,Rely)
;s(data,Data) ;s(cplx,Cplx) ;s(ruse,Ruse)
;s(docu,Docu) ;s(time,Time) ;s(stor,Stor)
;s(pvol,Pvol) ;s(acap,Acap) ;s(pcap,Pcap)
;s(pcon,Pcon) ;s(aexp,Aexp) ;s(pexp,Pexp)
;s(ltex,Ltex) ;s(tool,Tool) ;s(site,Site) ;s(sced,Sced)
):-
    scores(Prec,Flex,Arch,Team,Pmat,Rely,Data,Cplx,
            Ruse,Docu,Time,Stor,Pvol,Acap,Pcap,Pcon,
        Aexp,Pexp,Ltex,Tool,Site,Sced),

    vlvh(Prec), vlvh(Flex), vlvh(Arch), vlvh(Team),
    vlvh(Pmat), vlvh(Rely),  lvh(Data), vlxh(Cplx),
     lxh(Ruse), vlvh(Docu),  nxh(Time),  nxh(Stor),
     lvh(Pvol), vlvh(Acap), vlvh(Pcap), vlvh(Pcon),
    vlvh(Aexp), vlvh(Pexp), vlvh(Ltex), vlvh(Tool),
    vlxh(Site),!.
```

6 Start-up actions

Usual stuff.

```
:- sneak(
        ['defaults.omo' % see Figure ??
        ,'config.omo'   % see Figure ??
        ,ufp2sloc       % see §??
        ]).

:- commandLine.
:- ?verbose -> hello ; true.
```

7 OMO Support code

7.1 Multis/2

Fast, named, assertions

```
multis(Stuff,All) :-
    bagof(One,Stuff^multi(Stuff,One),All).

multi((Heads :- Tail),(Head :- Tail)) :-
    d2l(Heads,List),
    member(Head,List).
```

7.2 Fields/3

Poke some values into the named fields.

```
fields(Fields,Functor,Term) :- fields1(Fields,Functor,Term),!.
fields(_,_,[]).

fields1([],_,_).
fields1([Field|Fields],Functor,Term) :-
    fields2(Field,Functor,Term),
    fields1(Fields,Functor,Term).

fields2(Field,Functor,Term) :-
    clause(Field,(Term,_)),
    functor(Term,Functor,_),!.
fields2(Field,Functor,_) :-
    barph(badField(Functor is [Field])).
```

7.3 w/2

Convert scores to numeric weights

```
w(A,W) :-
    demand(s(A,S)),
    postArch(A,S,W),
    demand(num10(W)).

postArch(A,S,W) :-
    cocomo(When),
    lookUp(postArch(When,_),A,S,W).
```

7.4 Random types

7.4.1 Random strings

```
rsym(X)  :- nonvar(X),!.
rsym(X)  :- gensym(g,X).

rsym(_,X)  :- nonvar(X),!.
rsym(A,X)  :- gensym(A,X).
```

7.4.2 Random number within a range

```
rin(M,N,_,X) :- nonvar(X),!, number(X),M =< X, X =< N.
rin(M,N,O,X) :- Steps is integer((N-M)/O),
            between(1,Steps,_),
        Y is random(Steps+1),
        X is min(M + Y*O,N).
```

7.4.3 Random value of a list

```
rin(M,N,X) :- nonvar(X),!, number(X),M =< X, X =< N.
rin(M,N,X) :- Steps is integer(N-M),
          between(1,Steps,_),
          Y is random(Steps+1),
          X is min(M + Y,N).

rin(X,L) :- number(X),!, member(Y,L), X =:= Y.
rin(X,L) :- nonvar(X),!, member(X,L).
rin(X,L) :- length(L,N), rmember1(L,N,X).

rmember1([H],_,H)   :- !.
rmember1([H|T],N,X) :- Pos is random(N) + 1,
                       less1(Pos,[H|T],Y,L),
                       (X=Y
                       ; N1 is N - 1,
                         rmember1(L,N1,X)).
```

8 Knowledge base

8.1 Sample project

```
scores is [s(pmat,vl)
      ,s(pvol,l)
          ,s(ltex,l)
          ].

project is [cocomo(ga)
           ,label('eg#1')
           ,language(prolog)
           ,revl(10)
           ,newKsloc(100)
           ,adaptedKsloc(0)
           ,cm(0)     % new code
           ,dm(0)     % new code
           ,im(0)     % new code
           ,aa(2)     % basic module search + docu [4, p24]
           ,unfm(0.4) % somewhat familiar
           ,su(30)    % nominal value [4, p23]
           ,at(0)
           ,atKprod(2.4)
           ,scedPercent(100)
           ].
```

8.2 LOC per Function points

Also loaded, but not shown due to size, are tables showing producicitivity in different 482 different programming systems. It tables a *lot* of code to get anything done in binary, but less code as the language matures. So:

```
upf2sloc('1st generation default',320).
upf2sloc('2nd generation default',107).
upf2sloc('3rd generation default',80).
upf2sloc('4th generation default',20).
upf2sloc('5th generation default',5).
```

The units here are lines of code per function point. For more details, see Boehm.

9 Bugs

None known but many suspected.

Acknowledgements his research was conducted at West Virginia University under NASA contract NCC2-0979 and NCC5-685. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

1. S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.
2. C. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.
3. T. Mukhopadhyay, S. Vicinanza, and M. Prietula. Examining the feasibility of a case-based reasoning tool for software effort estimation. *MIS Quarterly*, pages 155–171, June 1992.
4. B. W.Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
5. J. Wielemaker. *SWI-Prolog*. Available from `http://swi.psy.uva.nl/projects/xpce/SWI-Prolog.html`.

A License

This software is distributed under the GNU General Public License.

A.1 nowarranty.txt

A.2 warranty.txt

A.3 conditions.txt

1. *prod1.pl* : "An example of the PROD Prolog delivery and documentation system."
   Available from `http://tim.menzies.com/pdf/03prod1.pdf`.
2. *prod.pl* : "A PROlog Documentation, and Delivery Tool".
   Available from `http://tim.menzies.com/pdf/03prod.pdf`.
3. *prod0.pl* : "TITLE": a bare-bones minimal example of PROD.
   Available from `http://tim.menzies.com/pdf/03prod0.pdf`.
4. *prodabout.pl* : "Motivations": the why and who of PROD.
   Available from Available from `http://tim.menzies.com/pdf/03prodabout.pdf`.
5. *family.pl* : "A family database": documentation of a very simple Prolog family database.
   Available from `http://tim.menzies.com/pdf/03family.pdf`.
6. *lib.pl* : "Commonly used predicates":
   Available from `http://tim.menzies.com/pdf/03lib.pdf`.
7. *cfg.pl* : "Handler for config files and command line options":
   Available from `http://tim.menzies.com/pdf/03cfg.pdf`.
8. *gpl.pl* : "Including GPL-2 in Prod":
   Available from `http://tim.menzies.com/pdf/03gpl.pdf`.
9. *omo.pl* : "Software cost estimation":
   Available from `http://tim.menzies.com/pdf/03gpl.pdf`.

Fig. 3 This document is part of the PROD delivery and documentation tool for Prolog applications. To find out more about PROD, the best place to start is memo #2.