

Function point based cost estimation for projects using Gedae

Andreas Bogner,
EADS Deutschland GmbH,
Defence Electronics,
89070 Ulm, Germany
++49 (0)731 392 4368
Andreas.Bogner@eads.com

Introduction

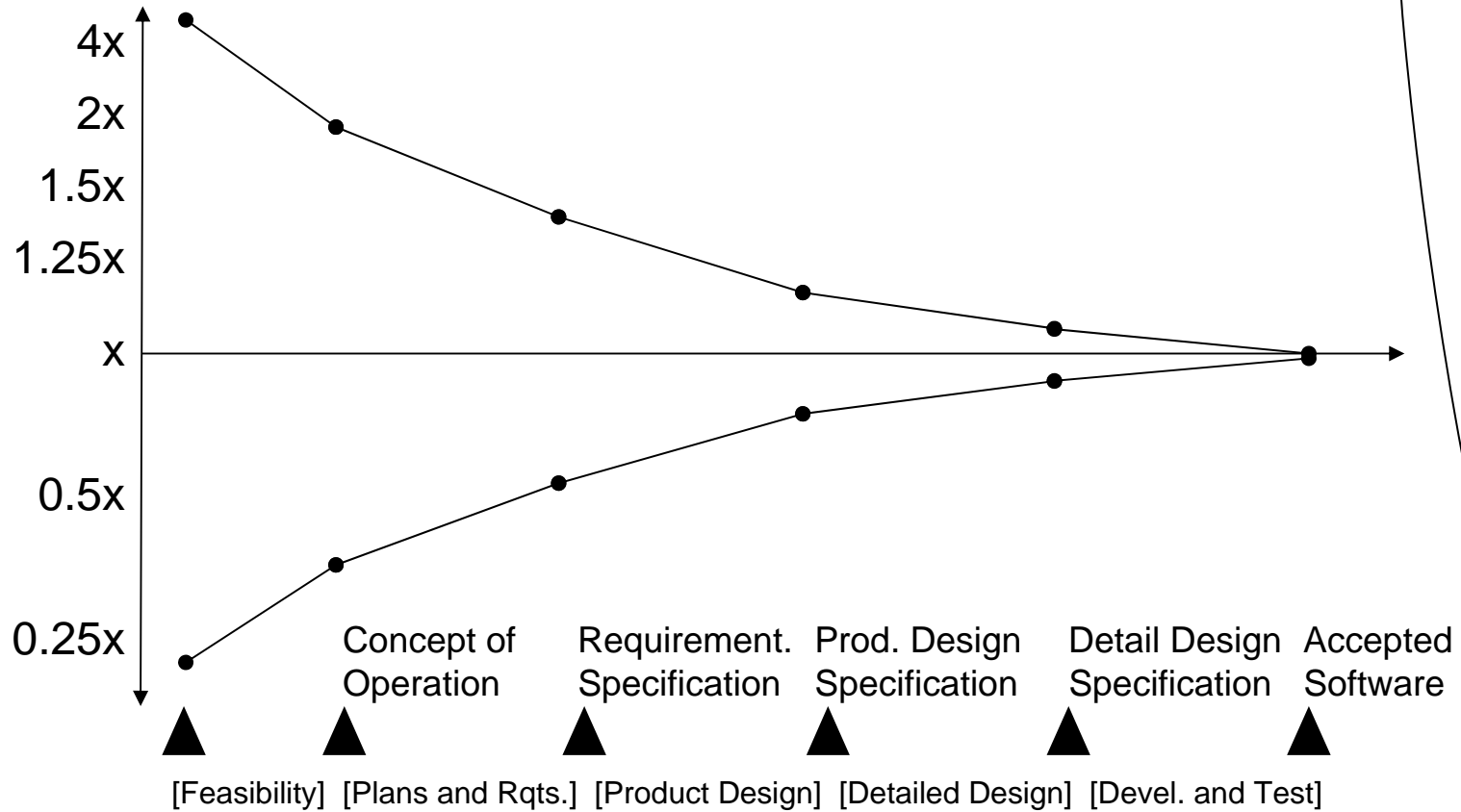
Barry W. Boehm said:

“Don’t become a statistic – take control of your software projects and plan for success!”

Overview

- Estimation fundamentals
- The traditional single line of code (SLOC)
- Basics on Unadjusted Function points (UFP)
- Parameters of UFP
- Comparability of UFPs and SLOCs
- General project adjustment factors
- Tools
- Conclusion

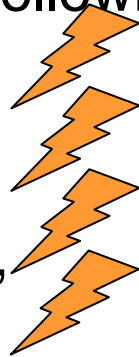
Unspecific fundamentals



Traditional single line of source code approach

If we want to estimate the effort by looking on the number of lines of the Gedae sources, we would need to estimate the following elements:

- primitives,
- flow graphs,
- parameter settings,
- group settings,
- user command program,
- and the system interface library.



This is the situation before Gedae generates the application.

Traditional single line of source code approach

If we concentrate on the elements after the application generation we would need to estimate the following elements:

- generated primitive functions,
- static and dynamic schedules,
- parameter files,
- user command program,
- and the system interface library.



These elements would need to be carefully investigated. Schedules for instance double, if the same processing is performed on two processors.

Basics of Unadjusted function points (UFPs)

- Function points measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types.
- The user functions have to be reflected each by a function point estimate.

UFP parameters are

- external inputs,
- external outputs,
- internal logical files (e.g. external states),
- external interface files (e.g. external database),
- and queries (e.g. look-up table, internal states, primitive with local data).

Every single count has to be rated low, average or high.
For instance, an input with less than for elements is rated low, more than 15 high and in between average.

Compare ability of UFPs to SLOCs

Programming language	UFP to SLOC
Machine code	640
Assembly (Macro)	320 (213)
ANSI-C	128
Fortran 77	107
Modula 2, report generator	80
Ada 83	71
C++	55
Java	53
Ada 95	49
Visual C++	34
Spreadsheet	6

Compare ability of UFPs to SLOCs

Prog. language generation	UFP to SLOC
First generation	320
Second generation	107
Third generation	80
Fourth generation	20
Fifth generation	4

A third generation language is meant to be processor, hardware and system architecture independent.

A Fourth generation language is system independent, modules can be encapsulated and reused.

A Fifth generation language generates the application directly from the user requirement.
(e.g. mathematical expressions).

General project adjustment factors

Scaling factors are:

- precedentedness,
- development flexibility,
- architecture,
- team cohesion
- and process maturity.

Adjustment factors are organized in:

- product
- platform
- personnel
- and project factors.

Product adjustment factors are

- required software reliability
- database size (Test data),
- product complexity,
- developed for reusability
- and match to life-cycle needs.

Platform adjustment factors are

- execution time constraint,
- main storage constraint,
- and platform volatility.

Personal adjustment factors are

- analyst capability,
- programmer capability,
- personnel continuity,
- applications experience,
- platform experience,
- and Language and tool experience.

Project factors are

- use of software tools,
- multi site development,
- and required development schedule.

Tools – USC-Cocomo II.2000

USC-COCOMO II.2000.0 - \DEIULMA.DE.NET.WORLDDH_A\$\ABOGNAN\home

File Edit View Parameters Calibrate Phase Maintenance Help

Project Name: CIP Signal Processin

Scale Factors

Precedentedness	NOM	3.72
Development Flexibility	NOM	3.04
Architecture / risk resolution	NOM	4.24
Team cohesion	NOM	3.29
Process maturity	NOM	4.68

OK Cancel Help

SLOC Input Dialog - EXAMPE1

Sizing Method

SLOC
 Function Points
 Adaptation and Reuse

Breakage
 % of code thrown away due to requirements evolution and volatility
 REVL: 20.00

Module Size in Function Points
 Language: USR1 Change Multiplier: 10

Function Type	# of Function Points			SubTotal
	Low	Average	High	
Internal Logical Files	0	0	2	30
External Interface Files	0	0	1	10
External Inputs	0	0	16	96
External Outputs	0	0	20	140
External Inquiries	0	0	5	30
Total Unadjusted Function Points				306
Equivalent Total in SLOC				3060

OK Cancel Help

R	h)	EA	Language	NOM Effort	EST Effort
1.00	1.00	USR 1		12.3	12.3

	Estimated	Effort	Sched
Optimistic	9.8	7.6	
Most Likely	12.3	8.1	
Pessimistic	15.4	8.7	

EAFF - EXAMPE1

base + Incr % = rating

Product: RELY DATA DOCU CPLX RUSE
 base: NOM NOM NOM NOM NOM
 Incr%: 0% 0% 0% 0% 0%

Platform: TIME STOR PVOL
 base: NOM NOM NOM
 Incr%: 0% 0% 0%

Personnel: ACAP PCAP PCON APEX LTEX PLEX
 base: NOM NOM NOM NOM NOM
 Incr%: 0% 0% 0% 0% 0%

Project: TOOL SITE
 base: NOM NOM
 Incr%: 0% 0%

User: USR1 USR2
 base: NOM NOM
 Incr%: 0% 0%

EAFF is also affected by Schedule
 EAFF: 1.00

OK Cancel Help

Conclusion

Cost estimation will always bear challenges, and never be without blind summits and difficulties.

The given approach mechanizes the estimation process and makes projects comparable on this level. In this way the experiences made with former projects help to refine the cost estimates of future projects.