



Impacts of Software Reuse

USC-CSE Focused Workshop
15th International Forum on COCOMO and
Software Cost Estimation

Rhoda Novak

JohnRhoda@msn.com

25-26 October 2000



Agenda

- Study Background
- Reuse sensitivity analysis
 - Methodology
 - Findings
- Recommendations



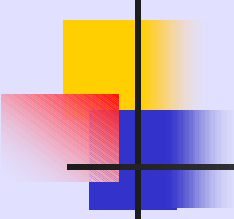
Acknowledgements

- Marilee Wheaton provided guidance on developing this methodology for using COCOMO II to bound software reuse cost estimates



Reuse Study for a Generic Space Program

- Programmatic
 - Early in program life
 - Challenging high-tech space program
 - State of art sensor
 - Intelligent, complex software
 - Immature / missing software requirements
 - Many operational issues unresolved
- Extensive software reuse planned
 - Reuse from other parts of the program
 - Reuse from other programs



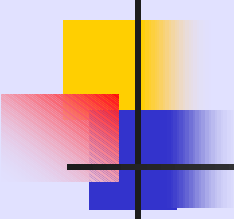
Typical Assumptions: Space System Developer

- Heritage programmers would rehost reuse code for this program
- Reuse candidates include nonoperational software
- Assumed operational software was reusable code appropriate for their program
 - Counted on heritage programmers to design the reuse code for both heritage and their program
 - Ignored the fact code was best effort, not designed for reuse



Study Overview : Issues

- Difficulty in estimating reuse when functions are in flux
- Program extension may impact availability of heritage programmers
- Reuse candidates may be harder to understand than assumed
- Code modification likely to increase



Approach to Bound Cost Estimation Uncertainties

- Since reuse code size estimation is in flux, use COCOMO to investigate “what if” impacts of
 - Fewer heritage programmers
 - Less understandable code
 - More modifications
- Bracket estimates with these factors
- Recommend risk reduction approaches
 - Provide metrics as early warning
 - Suggest program management approaches



Reuse Sensitivity Analysis

- Three typical reuse cases were selected
 - Reusable, high-quality and nominal
 - Each case had three “what if” conditions
 - COCOMO II reuse equations were used
 - Reference: COCOMO II Model Manual, updated 10-2-98, Adjusting for Reuse, pp.21-24 found under 1999 documents at <http://sunset.usc.edu/research/COCOMOII/index.html>
- NOTE: Reuse parameters should be tailored for the specific program being estimated



Baseline for Three Cases

- Reusable components
 - Highest quality, ideally suited for application, very easy to understand
- High-quality production code
 - High quality, well suited for application, easy to understand
- Nominal non-production code
 - Nominal quality, suited for application, nominal ease of understandability



“What If” Options

- Three “what if” options were analyzed
 - Baseline – described in findings
 - Options – only changes from baseline
 - Option a – decreased heritage programmers
 - Option b – increased in modifications
 - Option c – decreased heritage programmers and increased modifications



COCOMO II Reuse Parameters

- AAF – Adaptation Adjustment Factor (0 to 1)*
- SU – Software Understandability Increment (50 to 10)*
- AA – Assessment and Assimilation Increment (0 to 8)*
- UNFM – Programmer Unfamiliarity Increment (1 to 0)*
- ESLOC – Equivalent SLOC for reuse candidate
- ASLOC – Actual SLOC after reuse is implemented

* Values are listed from most optimistic to least optimistic



COCOMO II Study Parameters

	Case 1	Case 1a	Case 1b	Case 1c	Case 2	Case 2a	Case 2b	Case 2c	Case 3	Case 3a	Case 3b	Case 3c
DM	0	0	5	5	5	5	10	10	10	10	15	15
CM	5	5	10	10	10	10	20	20	20	20	25	25
IM	15	15	20	20	30	30	40	40	40	40	50	50
AAF (%)	6	6	11	11	14	14	22	22	22	22	28.5	28.5
ASLOC	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
AA	0	0	0	0	2	2	2	2	4	4	4	4
AAF	6	6	11	11	14	14	22	22	22	22	28.5	28.5
SU	10	10	10	10	20	20	20	20	30	30	30	30
UNFM	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6
ESLOC	62.4	67.2	114.4	123.2	171.2	193.6	257.6	292.8	286.4	339.2	359.2	427.6
% Increase within the same case	0	8	83	97	0	13	50	71	0	18	25	49
% Increase over case 1	0	8	83	97	174	188	125	138	359	405	214	247

UNFM – For this particular case where heritage programmers were scheduled to implement the reuse, it was assumed unlikely to get all heritage programmers and likely to get several heritage programmers

AA – It was assumed that increased assessment effort would be needed as the code went from reusable components to high quality to nominal. Without increased assessment, using less suitable components increases risk of size growth



Findings – Caveats and Notes

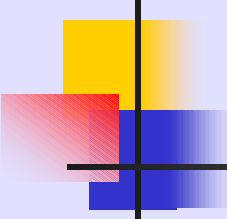
- CAVEAT: Quantitative findings apply only to the set of parameters selected for this study, but the general trends and methodology are of interest to a broad range of programs
- NOTE: Code growth is shown in percent of ESLOC growth for the implementation of reuse candidates, rather than computing actual cost growth. This permits the findings to apply to only the reuse baseline changes, rather than varying other cost estimation factors. ASLOC is assumed to be 1K.



Findings - Trends

- The most costly mistake is overestimating the quality, maturity, understandability and appropriateness of the reuse candidates
 - 174 to 359% between case 1 to 2 and 1 to 3 respectively*
- The impact of small modifications is larger when the baseline assumes almost no modifications will be needed
 - 83, 50 and 25% for cases 1 to 3, respectively*
- Fewer heritage programmers available increases ESLOC
 - 8 to 18% in this study for mostly familiar to considerably familiar*

* These values are for illustrative purposes and will change with the program's assumptions



Findings - Quantitative

	Baseline	Option a	Option b	Option c
Case 1 - Code Growth Within Case 1	0	8	83	97
Case 2 - Code Growth Within Case 2	0	13	50	71
Case 3 - Code Growth Within Case 3	0	18	25	49
	Baseline	Option a	Option b	Option c
Case 1 - Code Growth Within Case 1	0	8	83	97
Case 2 - Code Growth Compare to Case 1	174	188	125	138
Case 3 - Code Growth Compared to Case 1	359	405	214	247

Findings – Impact of Increasing Assessment Effort

	Case 1	Case 1a	Case 1b	Case 1c	Case 2	Case 2a	Case 2b	Case 2c	Case 3	Case 3a	Case 3b	Case 3c
DM	0	0	0	0	5	5	5	5	10	10	10	10
CM	5	5	5	5	10	10	10	10	20	20	20	20
IM	15	15	15	15	30	30	30	30	40	40	40	40
AAF (%)	6	6	6	6	14	14	14	14	22	22	22	22
ASLOC	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
AA	0	0	2	2	2	2	4	4	4	4	6	6
AAF	6	6	6	6	14	14	14	14	22	22	22	22
SU	10	10	10	10	20	20	20	20	30	30	30	30
UNFM	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6	0.2	0.6
ESLOC	62.4	67.2	82.4	87.2	171.2	193.6	191.2	213.6	286.4	339.2	306.4	359.2
% Increase within the same case	0	8	32	40	0	13	12	25	0	18	7	25
% Increase over case 1	0	8	32	40	174	188	132	145	359	405	272	312

Increasing Assessment & Assimilation (AA) only increases ESLOC by 32, 12 or 7% for cases 1 to 3, respectively, much less costly than misestimating code quality and understandability in the cost estimate baseline



Recommendations

- Provide metrics to track changes in baseline assumptions, so early corrective actions can be taken
 - Monitor code quality, modifications, heritage programmer availability
- Provide risk mitigation to support metrics
 - Consider potential risk reduction by increasing assessment effort for key modules
 - Cross-train heritage programmers on other modules