On Estimating Source Lines of Code from a Binary Program

Takahiro Sunada
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
takahiro-sun@is.naist.jp

Akito Monden
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
akito-m@is.naist.jp

Kenichi Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
matumoto@is.naist.jp

***Abstract*— Source Lines of Code (SLOC) is a most basic program size measure in software project management and/or quality assurance. This paper tries to estimate the source lines of code (SLOC) of a program from its binary code. In the proposed method, a binary program is disassembled, and library sections and data sections are removed. Then opcode frequency metrics are measured, and a multivariate regression model is built to estimate the SLOC. From an experiment with 23 C programs, our main result is that SLOC estimation from a binary program is possible, at least, in a limited environment. Our estimation model showed high accuracy in goodness of fit (R^2=0.928, MAE=14.1 and MMRE=10.4%).**

Keywords - program size measurement; binary code analysis; reverse engineering

I. INTRODUCTION

Source Lines of Code (SLOC) is a most basic and widely-used program size measure in software project management and quality assurance [1]. SLOC-based measures such as defect density (defects per SLOC), test case density (test cases per SLOC) and productivity (SLOC per person-hour) are commonly used in practice.

However, in recent software development, there are cases where SLOC is not available. One of typical cases is testing of a game program that runs on a consumer gaming console. It is a common situation that software test companies are forced to conduct testing of a pre-release version of a game program only having its executable binary code, i.e. no source code or other documents/deliverables available. This means that the size of the product to be tested is unaware. In such a case, quality assurance becomes extremely difficult due to lack of SLOC-based metrics such as defect density and test case density. This situation often happens to other types of software when outsourcing of software test takes place.

This paper tries to estimate SLOC of C programs from binary executables by a reverse engineering technique.

II. PROPOSED METHOD

As shown in Fig. 1, the proposed method consists of four steps.

[Step 1] Disassemble

A binary program is disassembled, and its assembly code is obtained.

[Step 2] Section analysis

Assembly code is analyzed to identify static linked libraries that are out of scope of SLOC estimation. Also, data sections are identified and ignored since it does not contain program code. Only code sections are analyzed in the next step.

[Step 3] Opcode analysis

In this step, types of opcode that have high correlation with SLOC are selected, and their quantities (we call

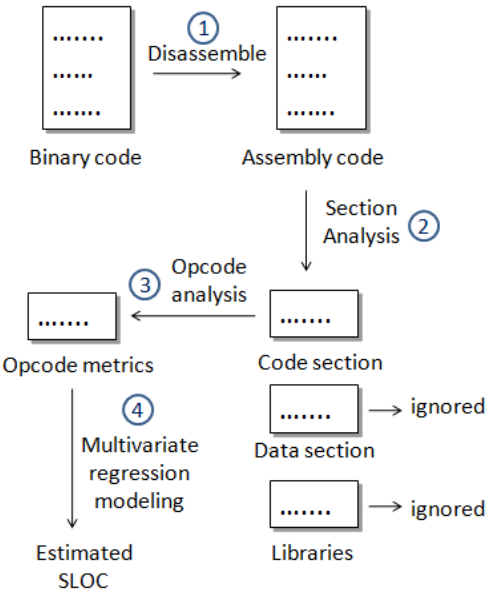

Figure 1. Overview of the proposed method.

TABLE I. Characteristics of 23 programs.

P	Spec	SLOC (physical)	SLOC (logical)	# of opcodes in code section
P_1	Blackjack	211	145	625
P_2	Bowling	168	115	551
P_3	Bowling	242	160	642
P_4	Coin game	144	86	432
P_5	Factorization	284	189	714
P_6	Factorization	255	125	769
P_7	Factorization	229	166	696
P_8	Arithmetic word	272	193	823
P_9	Arithmetic word	331	211	862
P_{10}	Arithmetic word	386	241	885
P_{11}	Hangman	181	151	371
P_{12}	Huffman	166	117	389
P_{13}	Huffman	127	79	413
P_{14}	Huffman	135	89	474
P_{15}	Huffman	158	94	348
P_{16}	Huffman	170	138	970
P_{17}	Coin game	250	200	964
P_{18}	Maze	139	84	310
P_{19}	Maze	331	214	858
P_{20}	Maze	402	336	2021
P_{21}	8-puzzle	415	264	1045
P_{22}	Bowling	196	119	666
P_{23}	Dice game	138	107	326

opcode frequency metrics) are measured.

[Step 4] Multivariate regression modeling

Using opcode frequency metrics as predictor variables, SLOC is estimated by multivariate regression modeling.

III. EXPERIMENT

A. Overview

In this section, we experimentally identify types of opcodes that have high correlation with SLOC, which needs to be selected in Step 3. Also, we demonstrate how opcode frequency metrics contribute to SLOC estimation in Step 4.

B. Materials

To identify a set of opcodes that can be used to estimate SLOC, we used 23 C programs each built from one of 10 functional specifications. These specifications include tiny games with command line interface such as Blackjack, Hangman, Hit-and-blow, 8-puzzle and Arithmetic Word Problems, etc. Programmers are master course students of Nara Institute of Science and Technology (NAIST).

All the programs were compiled by GCC version 4.3.4 (without optimization) in Cygwin environment [2], and obtained .exe files in PE format. Afterwards, all .exe files were disassembled by Diswin [3].

Table 1 shows characteristics of 23 programs P_1, ... P_{23}. Table 1 shows physical and logical SLOC, both commonly used in practice. Since physical SLOC is greatly influenced by the programming style, this paper uses logical SLOC, which counts the number of statements rather than lines of code. The logical SLOC of these programs ranges from 79 to 336.

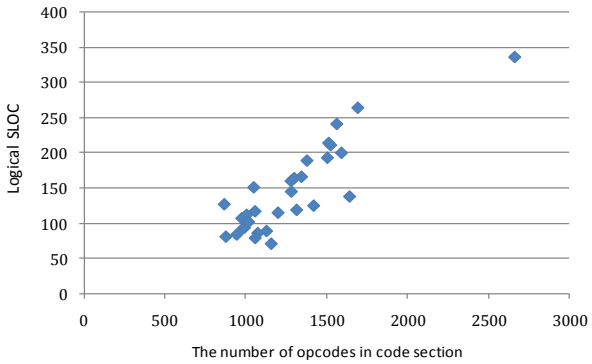

Figure 2. Overview of the proposed method.

The right most column of Table 1 shows the number of opcodes in code section of assembly programs; and, the relationship with logical SLOC is shown in Fig. 2. There are moderate relationship between logical SLOC and the number of opcodes. The coefficient of determination R^2 was 0.765.

C. Opcode Analysis

The result of the opoode (frequency) analysis is shown in Table 2. The bottom line of Table 2 shows R^2 values

TABLE II. Result of opcode frequency analysis.

P	mov	push	sub	test	call	and	jz	jmp	lea	leave	jnz	pop	cmp	jng
P_1	482	29	40	41	76	6	30	62	65	16	11	14	36	16
P_2	471	29	23	30	48	3	28	55	56	12	25	17	37	6
P_3	455	32	26	29	62	3	26	74	57	15	34	18	39	4
P_4	408	36	40	30	49	3	23	46	51	15	5	20	14	6
P_5	484	41	61	30	71	3	28	71	54	20	10	22	48	16
P_6	642	47	42	29	55	3	28	52	83	14	11	32	25	10
P_7	597	34	45	37	71	3	28	59	39	19	22	16	23	3
P_8	557	40	53	35	70	6	35	73	56	16	16	20	43	9
P_9	535	36	44	28	59	6	25	86	59	20	54	17	75	10
P_{10}	580	41	64	37	89	6	35	82	51	20	14	19	59	26
P_{11}	384	30	24	31	78	5	24	58	35	16	16	13	20	4
P_{12}	396	36	27	33	63	3	29	48	51	16	10	18	15	3
P_{13}	337	26	48	27	36	4	24	43	89	12	5	14	16	7
P_{14}	400	31	24	31	51	3	23	50	52	13	7	17	18	11
P_{15}	358	34	36	27	44	3	26	45	47	20	5	13	20	3
P_{16}	548	31	67	30	59	3	23	54	240	12	9	15	22	11
P_{17}	732	38	42	44	45	4	33	59	72	17	14	22	32	16
P_{18}	290	32	29	29	43	5	29	45	48	18	5	15	19	7
P_{19}	583	42	35	36	90	9	41	73	51	18	14	25	48	17
P_{20}	962	32	39	68	48	5	75	96	158	16	37	16	76	16
P_{21}	743	54	42	36	78	4	47	75	58	23	12	31	50	13
P_{22}	481	30	35	30	83	7	35	65	59	13	29	15	62	7
P_{23}	326	29	23	27	56	5	29	55	36	15	15	14	31	10
R^2 with SLOC	0.88	0.09	0.14	0.70	0.02	0.04	0.71	0.59	0.31	0.03	0.21	0.08	0.48	0.28

between the number of each opcode and logical SLOC. In the Table, 4 opcodes "mov", "test", "jz" and "jmp" had R^2 greater than 0.5. In particular, "mov" opcode showed the highest R^2 (0.88).

D. Estimation of SLOC

Using 4 opcodes frequency metrics as predictor variables, we carried out multivariate (linear) regression analysis to estimate logical SLOC. Stepwise variable selection was used to build a regression model. The following equation is the resultant model.

$$\textbf{SLOC} = 0.1868*\textbf{MOV} + 2.7231*\textbf{JMP} - 106.7546 \quad (1)$$

where MOV and JMP are opcode frequencies of mov and jmp instructions respectively.

In this model, opcodes "test" and "jz" were not selected as predictor variables.

Fig. 3 shows the result of estimation. X-axis is the estimated SLOC by the model, and Y-axis is the actual logical SLOC. Table 3 shows the goodness of fit of this model in terms of R^2, Mean Absolute Error (MAE) and Mean Magnitude of Relative Error (MMRE). Since the R^2 value greatly improved from 0.765, we consider that the model is much more useful in estimating SLOC than just counting the total number of opcodes.

TABLE III. Goodness of fit of SLOC estimation model.

R^2	MAE	MMRE
0.928	14.1	10.4%

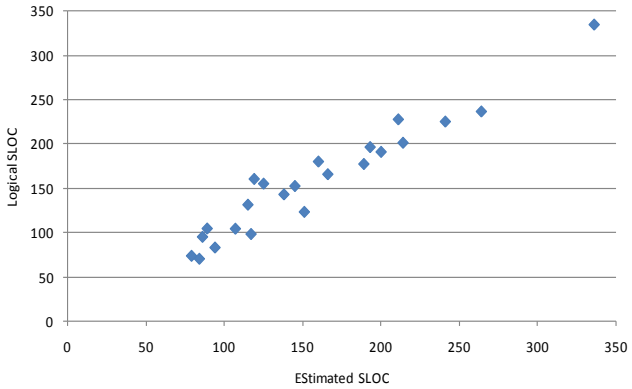

Figure 3. Result of estimation.

IV. CONCLUSION AND FUTURE WORK

In this paper we tried to estimate the logical SLOC of C programs from their binary executables by opcode frequency metrics and regression modeling. Our main result is that SLOC estimation from a binary program is possible, at least, in a limited environment. Our estimation model showed high accuracy in goodness of fit (R_2=0.928, MAE=14.1 and MMRE=10.4%).

However, this is just an initial stage of our research, and we have a lot of things to do in the future work as follows:

- Investigation of the effect of optimization in compiling. Since the optimized binary program may have different characteristics, we need to clarify how optimization affects the SLOC estimation.
- Experiment with different compilers. In this paper we used only GCC in Cygwin environment. However, we may have different SLOC estimation model for different compilers.
- Building a guideline of removing library code from a binary program. Detecting and removal of library code is not an easy task. We will need to investigate more programs and seek for some guideline to do this.
- Experiment with larger programs. In this paper we used small student programs. We will need to conduct an experiment with larger programs.

ACKNOWLEDGMENT

A part of this work was conducted in the StagE Project, the Development of Next Generation IT Infrastructure, supported by Ministry of Education, Culture, Sports, Science and Technology. Also, a part of this work was conducted under Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C) (22500028).

REFERENCES

[1] J. H. Baumert, “Software Measures and the Capability Maturity Model,” Technical Report of Software Engineering Institute, Carnegie Mellon University, no. CMU/SEI-92-TR-25, 1992.

[2] Cygwin project, http://www.cygwin.com/

[3] Diswin, http://www.vector.co.jp/soft/win31/prog/se011061.html