Cost Effective Software Engineering using Program Slicing Techniques

[1]Muhammad Saleem
Department of Computer Science and Engineering, Hanyang University Ansan Campus, S. Korea
0082-10-25146291
engr_saleemwazir@yahoo.com

[2]Rasheed Hussain
Department of Computer Science and Engineering, Hanyang University Ansan Campus, S. Korea
0082-10-23492659
rasheed1984@gmail.com

[3]Yasir Ismail
Department of Electrical and Electronics Engineering, Hanyang University Ansan Campus, S. Korea
0082-10-80626291
marwat_telecomm@yahoo.com

[4]Shaikh Mohsin
Department of Computer Science and Engineering, Hanyang University Ansan Campus, S. Korea
silent_touch2210@yahoo.com

ABSTRACT
Software Development is a complex and multidimensional task. Often software development faces serious problems of meeting key constraints of cost and time. Big projects which are well planned and analyzed, can end up in a disaster because of mismanagement in cost estimation and time allocation. Program slicing has unique importance in addressing the issues of cost and time. It is broadly applicable static program analysis technique which provides mechanism to analyze and understand the program behavior for further restructuring and refinement. In this paper, authors investigate the relationship between program slicing and software development phases on the basis of empirical studies conducted in the past and also establish the fact that how program slicing can be helpful in making software system cost and time effective.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management– *software cost and time.*

General Terms
Management, Measurement, Performance, Experimentation

Keywords
Program slicing, software cost and time, software development phases, program code

ICIS 2009, November 24-26, 2009 Seoul, Korea

1. INTRODUCTION
In the modern era of technological revolution, software development has emerged as backbone of business and operational world. Software systems serve as economical tools for multinational companies, infrastructural operations, and government dealings.

However, 40 years of research and innovation in software development, estimation of software development cost and time remain one of the most vexing issues in software engineering. Previously, Quantitative Software Management and Software Project Management have explored different approaches of safe and simple software cost analysis. But still an effective solution was prime requirement.

According to study, Testing, Debugging, Maintenance, Re-Engineering consumes more than half of labor and cost of overall software budget and resources. Program slicing has been recognized as an economical and efficient mechanism to optimize above mentioned software development levels [1]. Program slicing is a technique whose application is aimed at extracting out program parts that may not have computational influence on other statements. Purposefully program slicing focuses to identify semantically meaningful portion of a program on the basis of slicing criterion. It also works as program analysis tool by developing subset of inter-relevant large code program which is scattered through out the program. Consequently it helps software engineers to carry out all calculated operations on domain of interested program code.

Ongoing research of almost 25 years has evaluated and evolved many aspects of program slicing. From technical modification of large programs to code processing application Program Slicing , has emerged as vital component in restructuring and refinement of program code [2].The vital application of program slicing is recognized in program debugging, testing, software maintenance, safety, re- engineering [3].

In this paper, we have redefined program slicing application in modern software engineering phases. Previously

different approaches adopted in optimization and analysis of program code. But program slicing is far better choice than conventional techniques in terms of saving cost and time. We have carried out an analysis of program slicing effectiveness on the basis of empirical studies conducted in past [4], experimental results [5] and theoretical conclusions.

This paper is further organized as follow. Section 2 briefly explain overview of the program slicing, Section 3 describes application of program slicing in software debugging, Section 4 evaluates relationship between program slicing and testing on the basis of cost and labor consumption, section 5 shows maintenance phase redefined with program slicing applications in empirical and economical terms. Two other aspects of Maintenance are also elaborated.

2. PROGRAM SLICING OVERVIEW

Program slicing is a technique for simplifying programs by eliminating those parts of the program which are not of interest. The main aim of program slicing is to identify and extract relevant parts of a software program from a more complicated code. A slice is a reduced program which preserves the original program's behavior for a given set of variables at a chosen point of interest, referred to as slicing criterion. Typically, slicing criterion consists of pair (line no, variable).

The process of program slicing is especially important to software engineering efforts, as it allows the software engineer to find the way through complicated strings of code in order to access the source code that actually drives the application. Being able to extract these subprograms and view the source code makes it possible to identify a wide range of potential bugs and thus make the software run with more efficiency [8]. Figure 1, 2 shows an example of a simple slicing.

Original Program

1. a = 3;
2. b = a + 2;
3. c = 5 + a;
4.

Figure 1. Original Program for slicing

liced Program

1. a = 4;
2.
3. c = 5 + a;
4.

Slice wrt. (4; {c})

Figure 2 . Sliced Program

The program is sliced at with line 4 while variable "c" is used as slicing criterion. There are two basic methods for computing slices.

2.1 Static Slicing

In static slicing only statically available information is used for computing slices, hence called static slicing. Based on the original definition of Weiser, informally, a static program slice S consists of all statements in program P that may affect the value of variable v at some point p. The slice is defined for a slicing criterion C=(x,V), where x is a statement in program P and V is a subset of variables in P. A static slice includes all the statements that affect variable v for a set of all possible inputs at the point of interest (i.e., at the statement x). Static slices are computed by finding consecutive sets of indirectly relevant statements, according to data and control dependencies. Figure 3, 4 shows an example of static slicing

Original Program

1. int i;
2. int sum = 0;
3. int product = 1;
4. for(i = 0; i < N; ++i) {
5. sum = sum + i;
6. product = product *i;
7. }
8. write(sum);
9. write(product);

Figure 3. Original Program

This new program is a valid slicing of the above program with respect to the criterion (write(sum),{sum}):

Sliced Program

1. int i;
2. int sum = 0;
3. for(i = 0; i < N; ++i) {
4. sum = sum + i;
5. }
6. write(sum);

Figure 4 . Sliced Program

2.2 Dynamic Slicing

In the case of dynamic program slicing, only the dependences that occur in a specific execution of the program are taken into account. A dynamic slicing criterion specifies the input, and distinguishes between different occurrences of a statement in the execution history; typically, it consists of triple (input, occurrence of a statement, variable).

An alternative view of the difference between static and dynamic slicing is that dynamic slicing assumes a fixed input for a program, whereas static slicing does not make assumptions regarding the input. Figure 5 shows the difference between static and dynamic slicing.

Static Slicing	Dynamic Slicing

Function f(N)
1: 1: z = 0
2: a = 0
3: b = 2
4: p = &b
5: For i = 1 to N
{
6: If (i % 2 = = 0)
{
7: p = &a
}
8: a = a + 1
9: z = 2 * (*p)
}
10: Print(z)
	For input N=1
	1: z = 0 [z=0]
	2: a = 0 [a=0]
	3: b = 2 [b=2]
	4: p = &b [p=&b]
	5: For i = 1 to N [i=1]
	{
	6: If (i % 2 = = 0) [false]
	8: a = a + 1 [a=1]
	9: z = 2 * (*p) [z=4]
	}
	10: Print(z) [z=4]

Figure 5. Dynamic Slicing

So the result of the static slicing Static Slice (<10, z>) = {1, 2, 3, 4, 7, 8, 9, 10} while Dynamic Slice (<input = 1, variable = z, execution point = 10>) = {3, 4, 9, 10}

3. DEBUGGING AND PROGRAM ANALYSIS

Debugging is considered as major issue in software evolution. Debugging large and complex software systems evolved, requires a lot of effort since it is very difficult to localize and identify the faults. Conventional debugging process involves allot of overhead. Consequently software faults are not completely removed. This is because debugging is subject to software's which are often modified to reflect new functionality. Reducing the effort of debugging process is an important measure in efficient evolution of software.

Program slicing is very sound and promising approach to localize faults efficiently. Detection of faults in software product was initial motivation of program slicing. In debugging, software engineers focus their interest in specific execution of program. Program Slicing assist software engineers to develop comprehensive analyzer and focus their attention on those statements that contribute to a fault of commission.

The general steps for debugging are as follow;

First translate the external symptoms of the program failure into the corresponding internal symptoms in terms of data or control problems in the program. After that, one of the internal symptoms is selected slicing criterion for dynamic slicing. Using this criterion, a dynamic slice is obtained containing code that caused failure. After examining the dynamic slice, a statement is selected at which to examine the program state, and the program state is restored to the state when the control last reached that statement. Now the values of some variables are observed in the restored state to find fault. If search is not successful then user may choose to further examine the restored state, guess a new fault, or select a new slicing criterion and repeat the cycle until the fault is localized.

Experimental results and analysis of Kusumutu et al. [9] are shown in tables 1, 2. Table 1 shows the size of the program and of the slice obtained by the typical slicing criterion for failure of the program. Also, the tables include the distance (number of lines) from the statement at which the incorrect values are printed, to that at which the fault is localized. P1 to P8 are different programs for which various faults (F1 to F7) are localized.

Table 1. Size of program and slice

	P1	P2	P3	P4	P5	P6	P7	P8	Avg
LOC	427	432	428	428	429	429	434	434	430
Slice Size	194	194	196	196	199	199	202	211	199
Distance	110	111	120	5	131	21	47	16	_

From the table, the average ratios of the slice to the entire program are 46%. The extracted slice in each trial was reasonably small compared to the entire program.

Table 2 show the data about the time (in minutes) required to localize each of the faults in each trial.

Table 2. Fault localization results with and without slicing

	try	F1	F2	F3	F4	F5	F6	F7	total
Without Slicing	1	17	10	26	27	35	17	7	139
	2	28	18	36	17	25	23	16	163
	3	23	12	28	32	41	17	7	160
With Slicing	1	38	6	27	18	20	20	8	119
	2	11	8	10	16	16	13	7	88
	3	14	14	19	36	10	5	17	116

The average time to localize all seven problems without using program slicing is 154 minutes while it is 107.66 minutes if we use program slicing.

Practically program slicing optimizes code up to 25% of code analyzed by conventional tools. Notable time difference was also observed.

4. SOFTWARE TESTING

Testing is an important phase in process of software development and Re-engineering. Testing insures error free software. Efficiency and Accuracy of test –cases are subject certain issues. Testing consumes at least half of labor expended to produce a working program. Power and utility of program slicing comes in assisting software engineers to adopt unique mechanism of testing. The process of negotiating a software testing budget can be complex task. Particularly automation of the testing process should be beneficial because the typical testing process is human–intensive activity and generally costly, time consuming, error prone and inadequately done [6]. Slicing a portion of program and extracting scattered statements which are relevant to particularly added or new functionality would help make analysis and catch bugs. Influential application of program slicing was observed in Regression Testing, and Incremental Regression Testing which manages re-

testing of program after a modification made into it. Goal is to insure that bug fixes and new functionality do not adversely affect original program semantics or syntax. Testing carried out using program slicing techniques observed following differences with conventional testing approaches as show in table 3.

Table 3

Comparative analysis of average testing and Program slicing based testing		
C program segment	413 tests carried out.	
14,500 LOC	Average test execution	Execution using program slicing mechanism
6500 basic blocks	11%	71%
183 functions	26%	88%

Above calculated results suggests that testing can be regarded as an investment for software project. Program slicing with its application can enable software to generate revenue [7].

According to hypothetical case study, cost of quality technique to analyze return on the testing is carried out. Analysis is further formatted in table 4.

Table 4. Testing Investment Analysis

Testing	Manual Testing	Using Program Slicing
Fix-Bugs , Internal Failure investment	3500$	500$
Return on Investment	350%	445%

According to another experimental study conducted by David Brinkley, and Mark Harman , over 1000,000 lines of code was put to testing using program slicing. Impact of computation time in that experiment illustrates 71% reduction in run-time and 64% reduction in the memory usage.

Brinkley further described an algorithmic approach to reduce the cost of regression testing using program slicing. (i) Reducing the number of test that must be re-run (ii) Decreasing size of the program that would be subject to some test. This is accomplished with slicing technology and its powerful impact of lines of code.

5. SOFTWARE MAINTANENCE

Software maintenance is phase of software engineering in which necessary modification into software product is made to remove faults, to improve its efficiency and performance, to enable the product as adaptable to changing environmental Factors.

Over the years software engineers have realized that maintenance costs are important part of over all software cost and crucial to success of software project. According the software development giants, software maintenance consumes 67% of software life-cycle cost.

Program slicing is applied to software maintenance problem by extending the notion of program slice to a decomposition slice. Decomposition slice captures all computation on a given variable or criterion. Thus program slicing facilitates the maintainers with a technique of identifying those statements which are subject to modification or what are not. Since slices produced are semantically consistent and can be merged back into original program after some operation performed on them.

We further explain two important aspects of software maintenance: comprehension and software re-engineering with notion of program slicing application into them.

5.1 Software Re-Engineering

Software Re-engineering is process of constantly updating and renovating business-critical software systems. This is hardware level aspect of Maintenance which comes in practice because of business requirement change, technological infrastructure and other external factors. Re-engineering is costly process. Some systems are beyond repair. In such a situation, a re-engineering process needs to be undertaken. Even where systems are not beyond repair, it is often useful to be able to extract components of the systems for reuse and re-structuring. Slicing has been used as a component in strategies for restructuring [7] Re-engineering. This will optimize the re-engineering process and will reduce cost and time overheads by focusing on particular slice for change or innovation.

5.2 Software Comprehension

To maintain the quality of software high and reliable depends on understanding of system code by experienced software professionals. Comprehension is process of understanding semantics and syntax of program code. This is vital part of maintenance process which can lead to develop effective software matrices. Comprehension can allow engineering to make an assessment of human performance in software development process and improve the understandability of code. Comprehension is primary phase of software maintenance which cost about 50% to 90% of the overall maintenance budget. Complexity of comprehension arises when dealing with large code. Large-scale study of 43 c- programs was conducted to reflect difference of code understanding using program slicing. Table 5 is data collection of those c-programs.

Table 5

Program	LOC without comments and blank lines	No of lines after applying slicing	Percent reduction in No of lines	Affect of comprehension assisted by slicing on over all software budget, time
a2ps	40,222	12,348	30.7%	15%
acct-6.3	6,764	501	7.4%	3.6%
barcode	3,975	1,212	30.5%	14.9%
bc	11,173	5,452	48.8%	23.9%
byacc	5,501	996	18.1%	8.9%
cadp	10,620	828	7.8%	3.8%
compress	1,431	356	24.9%	12.2%
copia	1,112	252	22.7%	11.1%
csurf-pkgs	38,507	6,007	15.6%	7.6%
ctags	14,298	5,948	41.6%	20.4%
cvs	67,828	31,404	46.3%	22.7%
diffutils	12,705	2,871	22.6%	11.1%
ed	9,046	4,822	53.3%	26.1%
empire	48,800	16,104	33.0%	16.2%
EPWIC-1	5,719	646	11.3%	5.5%
espresso	21,780	6,708	30.8%	15.1%
findutils	11,843	3,257	27.5%	13.5%
flex2-4-7	10,654	2,642	24.8%	12.2%
flex2-5-4	15,283	3,194	20.9%	10.2%
ftpd	15,361	5,361	34.9%	17.1%
gcc.cpp	5,731	2,625	45.8%	22.4%
gnubg-0.0	6,988	1,558	22.3%	10.9%

Data suggests that with decrease in lines of code obtained there was consequent decrease in budget of overall software. Such enormous percentage decrease in code analysis allows software engineers to re-structure and refine the program feasible enough for program slicing. This sample data is evidence of computational effect produced by program slicing in code processing theory.

6. CONCLUSION

The prime objective of our work is to analyze the experimental results, empirical studies, hypothetical calculation based on program slicing. We have established the fact that program slicing mechanism can be standardized to be vital part in software matrices and economics. In future we look forward to focus on more experimental results to evaluate program slicing importance and role.

7. ACKNOWLEDGMENTS

This research work is sponsored by 'Higher Education Commission (HEC), Govt. Of Pakistan' under the scholarship program titled: MS Level Training in Korean Universities/Industry.

8. REFERENCES

[1] Mark Weisier. Program Slicing IEEE Transaction on software engineering, 1984.

[2] Baowen Xu ,Ju Qian Xi and Xiaofang Zhang . A Brief Survey of Program Slicing , ACMSIGSOFT Software Engineering Notes, 2005

[3] David Binkley and Mark Harman. Emprical Study of Optimization Techniques, Loyola College inMarylang, 1997

[4] O. Cetinkaya and D. Cetinkaya. An Empirical Study of Static Program Slice size , ACM Transaction on Software Engineering and methodology (TOSEM)

[5] Takashi Ishio and Shinji Kusumoto. Program Slicing Tool for Effective Software Evolution Using Aspect-Oriented Technique, Proceedings of the 6th International workshop on Principles of Software Evolution, 2003.

[6] David Binkely. Application of Program Slicing to Regression Testing, Information and Software Technology Issue on Program Slicing, 1999.

[7] A. Fujioka, T. Okamoto, and K. Ohta. Investing in Software Testing , Australia, pp. 244-251, 1992

[8] http://www.wisegeek.com/what-is-program-slicing.htm

[9] Shingi kusumoto, Akira. Experimental Evaluation of Program Slicing for Fault Localization, Empirical Software Engineering, volume 7, pages 49-76, 2002.