Auto Submission , Grading and Copy Checking of Assignments

A thesis submitted
in partial fulfilment of the requirements
for the degree of

Bachelor of Technology

by

Pankaj Khandelwal

to the

Department of Computer Science And Engineering

Indian Institute of Technology, Kanpur

April, 1996

CERTIFICATE

It is certified that the work contained in the thesis entitled *Auto Submission , Grading and Copy Checking Of Assignments*, by Pankaj Khandelwal has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Dheeraj Sanghi,
Associate Professor,
Department of Computer Science
and Engineering,
IIT Kanpur

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all the people who have helped me during the course of this work. Amongst these, Dr. Dheeraj Sanghi occupy a special place. As guide of my project he was always there to help me, to guide me and to encourage me when I was in low spirits. His dedication to work will always be a source of motivation to me. The choicest epithets could not suffice to express my gratitude for him.

I would also like to thank Mr. Nirmal Roberts for his help in implementing this package on hp system.

Contents

Chapter 1

Introduction

1.1 Motivation

In an academic environment, assignments are important to ensure proper learning and to give the students a feeling for practical applications of the theoretical concepts taught in the class.

The assignments are to be submitted to the instructor-in-charge (or the T.A.) for evaluation.Assignment submission by using electronic mail enforces unnecessary burden on the T.A. .The header is to be removed and the assignment is to be saved.This is an unnecessary overhead.What's desirable is that the students must be able to submit there assignments without this overhead to T.A.

A very important aspect of computer programming is *good* programming style. It is very important for the software professionals to instill good programming practices from the beginning.An automated tool is therefore required to evaluate the assignments on these so called good programming practices.

Anyone who has done computer programming is very well aware of the nuiscence of copying.This is a big headache for all the instructors, specially in situations where a large number of students are enrolled. Manual checking by comparing assignments is almost an impossible job.Here again an automated tool is required .

1.2 Aim

This BTP aimed at the development of a software package to automate

- Assignment Submission
- Grading and
- Copy Checking

The Automated Grading and Copy Checking tool is clearly programming language dependent.Pascal was chosen since this package is useful for the beginners and the course *Fundamentals Of Computing* , ESC101 is taught in Pascal.

Chapter 2

Software

2.1 On-line Submission Of Assignments

Once a student has developed a software-oriented class project, what are reliable ways for students to submit their programs? Simply opening up permissions a shared file system so that any student can place their files there is unsuitable: students would also be free to access (and potentially delete) files belonging to others. Using the host's electronic mail (email) system has its own problems: while not as public as the first approach, it has the disadvantage of requiring the instructor to manually save each message, and then edit it to obtain just the body of the message. This task is made more complicated when students have *many* different assignments for which they must send text.

A software has been developed to simplify this management task. In order to hand in an assignment electronically, students simply run a program called `submit`, and indicate which programming assignment their file fulfills. Our system then accesses the student's text, files it under that student's name in a unique directory dedicated for that assignment, and returns to the student a message confirming the safe transmission of the text. The central directory where all submissions are organized is not accessible to students.

This software contains features to increase our confidence in its security and reliability. The program performs a number of checks on the integrity of transferred data before printing a message to the student as a 'receipt.' The binaries are constructed and installed in such a way as to inhibit malicious users from changing them to exhibit undesirable behav-

ior .Specific roles are defined for the system administrator, instructor-in-charge , Teaching Assistants and the students ,in order to achieve above mentioned goal. The instructor can control the sizes and type of text that is submitted by students (as a protection against malicious users who might try to fill the directory with junk text and hence deny other students from submitting assignments.)

Another feature incorporated is regarding late submission of assignments. Submissions made after the due date are suffixed "LATE".

2.2 Grading On Programming Style

While implementing the software design of a system in any programming language, the programmer has to follow certain guidelines to make the source code easy to read and understand.

An attempt has been made to measure how far a programmer is following the specified guidelines.This section outlines a systematic procedure used to measure the programming style of a programmer and to grade him for the same.

Each program is given a percentage of *score* for style that consists of contributions in varying degrees from following program features:

Module length: The average length, in non blanks , of module definitions. A low figure implies a well structured, understandable program.

Identifier length: The average length, of user identifiers; This gives an approximate indication of the use of meaningful names. Readability increases with longer identifier lengths.

Comments: The percentage of characters used for commenting; Understandability of code increases with more comments.But after a certain point , excessive usage will obscure the program statements.

Indentation: The ratio of initial spaces to total number of characters; Here also a high

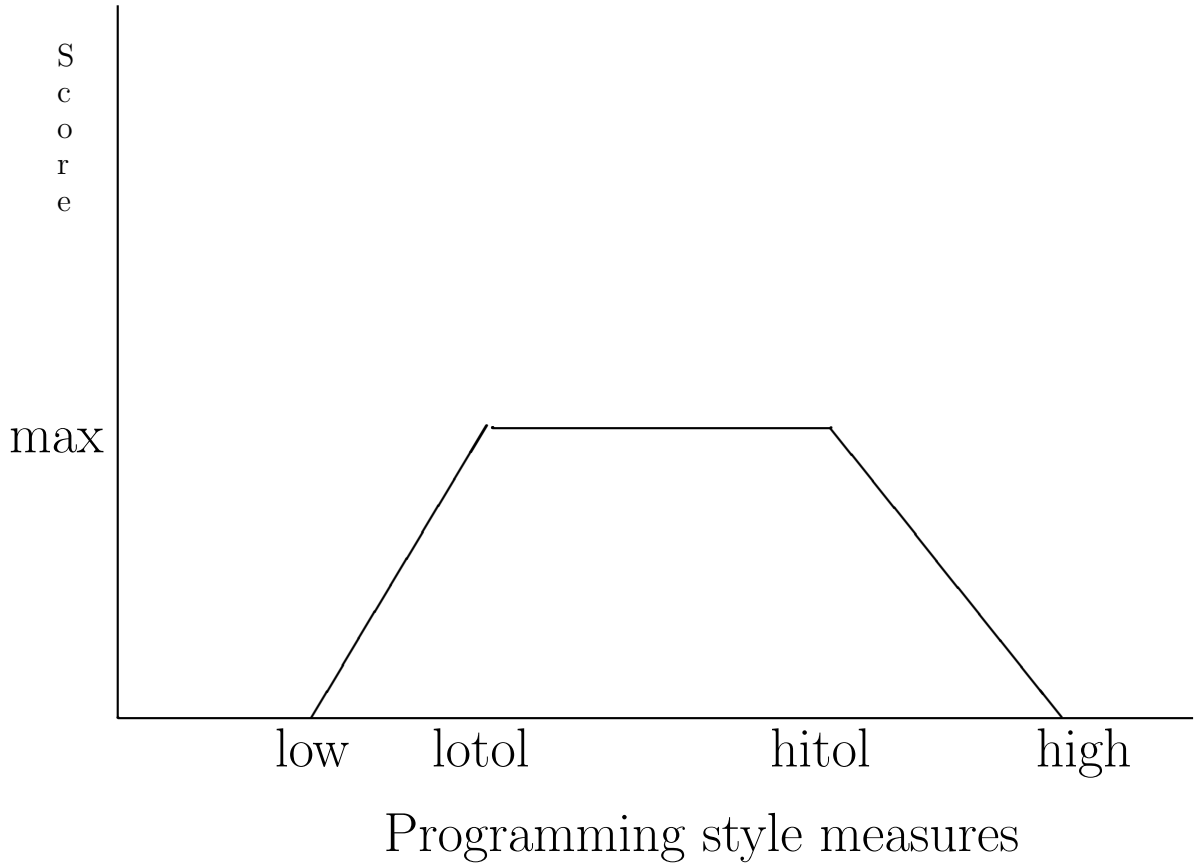

Figure 2.1: Metric Value Calculation

figure is taken to mean a better style.

Goto's: The number of occurances of a goto statement; A zero figure is assumed to be best style.

A score is associated with each metric defined above. Each contributes to a different maximum percentage to the final score, and each is additive , with the exception of the last which is subtractive.

For each metric , five parameters are provided which define a conversion curve as shown in fig 2.1.

METRIC	MAX	LO	LOTOL	HITOL	HI
Module Length	29	4	10	35	50
Identifier Length	27	3	5	10	14
Comment Chars	22	8	15	25	35
Indentation	22	8	24	48	60
Gotos	-20	1	3	99	99

Figure 2.2: Parameters of Programming style metrics

The parameter 'max' specifies the maximum percentage score to be given for each measure. If the value of measure lies in the range defined by lotol and hitol , then maximum score for that measure is given. Similarly, if measured value fall in either the ranges (low,lotol) or(hitol,high) then a linear proportion of the maximum score is assigned. When the value of the measure is less than low or greater than high, a score of zero is obtained. The scores for each measure are summed except that of goto's, which is subtracted from the total . The conversion graph shows that too high or too low figure for each metric decreases the final score, since these values indicate poor style.

The fig 2.2 provides values of parameters to each metric.

2.3 Copy Checking

Assignment copying has been the headache of almost all the instructor's, more so for the programming courses.If the number of students submitting the assignments is less, than manual checking is possible.But as the number of instances of an assignment increases , checking for copied assignments is extreamly difficult.A tool for identifying copied assignments was developed.

A small survey was done in order to identify the features of program that are changed by students before submitting the copied assignments.The finding of the survey was that students tend to change the programming style and the input output features.The features that were changed were:

- Identifier names
- Indentation
- Comments
- Input/Outout features
- Introducing trivial functions

With the help of above results certain characteristics of programs were identified which remain unchanged even after the above mentioned changes Were incorporated, except the introduction of trivial funtions.Following features remain unaltered:

- Source code metrics
- Data flow metrics

2.3.1 Source Code Metrics

Operators:
Operator is any symbol or keyword in program that specifies an algorithmic action . For example, arithmetic symbols,punctuations , semicolon, if then else, do until , while do, goto , names of subroutines ,and functions are operators in the program.

Operands:
Operand is a symbol used to represent data. Variables , literals , and labels are treated as operands.

Clearly, the number of occurances of each of these operators(All the Pascal keywords) and the number of operands remain unchanged for copied assignments.

It must be noted that certain changes do occur due to changed input output features and the additional trivial functions as obtained from the survey. The equality that we are talking about allows for some tolerances due to these changed features.The tolerance value for identifiers is 1 and that for identifiers has been set at 3.

If the number of occurances of an operator is found to be same for the two assignments under consideration, copy_probability is incremented by 1.This is done for each operator.

If the programs being compared have same number of variables then the copy_probability is incremented by 10.

The above mentioned procedure gives a copy_probability for the programs being compared out of a maximum of 105(95 for various Pascal keywords and 10 for identifiers).The default cut-off probability has been kept at 85, ie, if the copy_probability exceeds 85 marks then the programs are assumed to have been copied.

2.3.2 Data Flow Metrics

Metrics under this classification measure the flow of data within the program.

Live variables:
A variable is live from it's first reference to the last reference.Average live variables per statement is the sum of count of the live variables divided by the count of executable statements in a procedure.

Variable span:
Span is the number of executable statements between successive references to the same variable.For a program that references the same variable in n statements, there are (n-1) spans for that variable. Average span size is the average number of executable statements that pass between successive references of variable.

In the light of program features that are changed generally, we can easily see that data flow metrics of every procedure and function remain unchanged.These measures would change only when extra references to already existing variables are made or new variables are introduced. But such changes would invariably include algorithmic changes. In the case of copied assignments, such changes are generally not made and hence we can safely assume that they remain constant for copied assignments.

Average live variables and average span size were evaluated for all functions of the programs.The values of these metrics were indeed found to be very close for the copied

programs.

Once a set of pairs if assignments have been identified by the use of *Source code metrics* , a glance at the *Data flow metrics* could be used to discard some of the pairs that have been falsly implicated for copying.

Ideally, the comparisions among programs implicated by *Source code metric* evaluation for *Data flow metric* values should be automated.But this could not be done since I was stuck with how to compare the two *similar* procedures of two different programs.To compare Data flow metric values we need to identify the procedures for whom the comparisions would be done.So, only manual comparision is done for the time being but this itself is much better than manually comparing the two programs.

Chapter 3

Experiments

Experiments were conducted on five sets of assignments from the course *Fundamentals of Computing* in order to test the tools developed.

The aim of the experiments ranged from testing On-line submission tool for possible security flaws to empirically determining the values of various parameters used in copy checking.

3.1 On Assignment Submission

The binaries for On-line assignment submission were installed on Apah. The aim of the experiments was to test the programs for possible security flaws and ensure proper functioning of the installed binaries.

The details of implementations had to be worked out since system previlages were being used .

The results of the experiments were improved user interface , integrity checks on user's data ,unforseen errors were detected and corrected.

3.2 On Grading

The assignments of the sample sets were tested for the quality of grading achieved by the developed tool.The tool aimed at enforcing good programming style and to grade assignments based on the extent to which prescribed programming practices were followed.

Results:

The results obtained were highly satisfactory. For over 95% of assignments the marks obtained, were according to the student's programming style.The unsatisfactory results in rest of the assignments were because of the usage of average values for the evaluation of style metrics.For example:

- A program with a very large procedure and 5 small procedures was rated good since the average came out to be close to optimum value.
- A program was poorly indented, *for eg:* there was no alignment for the *if then else* statement but good marks were obtained since the student had left quite a lot of space in the beginning of statements.
- One of the program had no commenting except that a whole procedure was commented.Needless to say that good marks were given for commenting.

All the cases cited above are very difficult to handle, if not impossible.It's simply not possible to differentiate meaningful comments from the trash.But such cases were really in minority and can be treated as exceptions.

3.3 On Copy Checking

The experiments were the most important for copy checking since a number of parameters were required to be empirically determined, in order to enhance the reliability of this copy-checking tool.Reliability, itself was very important to be checked.

Determination of parameters:

First of all the tolerances for the source code metrics were determined.The values taken were those that gave most consistent results across the set of assignments.The tolerance

value for each Pascal keyword was taken as 1 and that for the number of identifiers was 3.

After determining tolerances, default cut-off copying probability was determined. This was set at 85 because of the following results:

- In the range (80-84) only one pair of assignment was found to be actually copied.
- Most of the assignments with copying probability above 90 were actually copied.
- In the range (85-90) no clear pattern across assignments emerged.

An important observation was that the optimum cut-off probability was highly dependent on the kind of assignment.For example, a simple program like *Hello World* program would be similar for almost all the student's and would therefore require a very high cut-off probability, whereas a very complex assignment would have a lot of dissimilarities, and would require a low cut-off probability.

In the case of current sample, we will consider *Matrix Operations* and *Chess*.The implementations for *Matrix Operations* would obviously be very similar whereas that for *Chess* program are expected to be dissimilar.Indeed this was the case obtained. With cut-off probability 85 the number of indicated copied assignments were 41 for *Matrix Operations* and 7 for *Chess*.Indeed 85 was the optimum cut-off for *Chess* and for *Matrix Operations* 90 was the optimum cut-off probability , with number of indicated copying dropping down to 5.

Results:

The tool developed is not foolproof but fairly reliable.It is possible that this tool may fail to indicate a copied pair or indicate a pair falsly. The importance of this tool is that it drastically reduces the number of programs to be manually checked for copying, thereby greatly reducing the work to be done to catch cheating.

An example to indicate the usefulness of this tool is in order.For the *Chess* assignment, 35 submissions were considered.Manual checking would have required 595 (35 * 17) comparisions of programs. With this tool , taking the default cut-off probability 7 pairs were indicated as copied. A glance at *Data Flow Metrics* discarded 3 pairs immediately. The remaining 4 pairs were indeed copied.

Chapter 4

Conclusion

In this project a software package has been developed to automate on-line assignment submission, grading on programming style and copy checking.

The binaries for On-line Submission have already been installed on Apah and is currently being used for the *Fundamentals of Computing* course.

A tool for grading and copy checking has also been developed. It was run on test programs and was found to give satisfactory results.

Chapter 5

Manuals

5.1 alsub

NAME

alsub - The user allows the students to submit assignments.

SYNOPSIS

alsub

DESCRIPTION

The user is prompted for the course_no for which he is allowing submissions to be made to him.

A directory "SUBMIT" is created in the user's home directory, with the courses of which he is the T.A. as sub-directories.

Each course has a sub-directory for each assignment.The number of assignments in a course is given by the instructor-in-charge.

WARNING

The user must be named as a T.A. for the *course* by the instructor. (See "tainfo")

The directory permissions and group must not be changed otherwise students will have problem in submitting assignments.

EXAMPLES

example% alsub

See Also

criclist tainfo submit

5.2 criclist

NAME

criclist - creates instructor course mapping for assignment submission.

SYNOPSIS

criclist

DESCRIPTION

Criclist creates the instructor course mapping required for assignment submission using "submit". The system administrator executes this binary to give the instructor's login name against the course-no that they are taking.

A faculty member can execute "tainfo" only after he has been named as instructor for the particular course.

OPTIONS

All the options are displayed in a tabular format on the execution of this binary.

EXAMPLES

example% criclist

See Also

tainfo alsub submit

5.3 submit

NAME

submit - Submits an assignment for the user.

SYNOPSIS

submit

submit -(l/s) course-no assign-no source (talog / section)

DESCRIPTION

Submit submits the assignment *assign-no* in the source file *source* for the course *course-no* .

If -l option is used , T.A.'s login name must be given.

If -s option is used , section number must be given.

The assignments submitted after the due date are suffixed LATE.(See "alsub").

If multiple submissions are done for the same assignment then the older ones are overwritten.

If an assignment consists of multiple files then first create archive files using "shar" and then submit.

OPTIONS

-l Assignment submission using T.A's login name.

-s Assignment submission using section number.

EXAMPLES

The user is prompted for all the inputs.
example% submit

To submit assign-no 2 of esc101 stored in fibonacci.p to T.A. pankh
example% submit -l esc101 2 fibonacci.p pankh

To submit assign-no 3 of esc101 stored in fibonacci.p to section a1
example% submit -s esc101 3 fibonacci.p a1

WARNING

The T.A. to whom assignment is being submitted must have allowed submission to be made.

See Also

criclist tainfo alsub

5.4 tainfo

NAME

tainfo - Allows the instructor to give information about the course he is taking.

SYNOPSIS

tainfo

DESCRIPTION

Tainfo is used by instructor to provide following information about the course:

- The number of assignments.
- Expected maximum size of an assignment.
- T.A. login names for the course.
- Sections, if any.

The user is prompted for all the inputs.

WARNING

A faculty member can execute "tainfo" only after he has been named as instructor for the particular course by the system administrator.

EXAMPLES

example% tainfo

See Also

criclist alsub submit

5.5 grcp

NAME

grcp - Automatic grading and copy checking for pascal programs.

SYNOPSIS

grcp -[g] -[c] dir

DESCRIPTION

grcp performs two functions for Pascal programs.

- Grading tool.
- Copy checking tool.

Grading tool:

grcp evaluates how far the programmer is following the programming guide lines. Program is given a percentage of "score" for style that consists of contributions in varying degrees from different programming guide lines that are to be followed by the programmer.

Programming features that are considered by this tool are identifier length, comments, indentation, usage of goto's, and module length. A score is associated with each feature said above. Each contributes to a different maximum percentage to the final score,and each is additive , with the exception of usage of goto's which is subtractive. Too high or too low usage of above features are detrimental to the final score since these indicate poor style.

The marks obtained for each program are stored in file "MARKS".Detailed marks for each programming feature is in style.out.

Copy checking tool:

Considers the similarity of two programs in terms of statistical measures and predicts the possibility of copying. The method is not foolproof but is fairly reliable. The user must check the two assignment indicated as similar lest innocent might get penalised.

The probable copied assignments are stored in file "TOPA".

FLAGS

-g grades the programs in the directory *dir* on programming style.

-c checks the programs in *dir* for possible copying.

EXAMPLES

To grade programs in directory TEST on style measures
example% grcp -g TEST

To check copying among programs in TEST
example% grcp -c TEST

To grade as well as check for copying in TEST
example% grcp TEST
example% grcp -g -c TEST

WARNING

grcp expects syntactically correct PASCAL programs. The results produced may be dubious, if improperly compiled PASCAL programs are given as input.

Bibliography

[1] Conte , Dunsmore , Shen. "Software engineering metrics and models".

[2] "Automatic assesment aid for Pascal programs". Michael J. Rees. ACM SIGPLAN,1981.

[3] "A Style Analysis of C Programs". Berry.R.E.,Meekings B.A.E. CACM Jan,1985.